FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Projektové řízení pomocí GitHub Projects
Student:	Adam Warzel
Vedoucí:	Ing. Josef Gattermayer, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

GitHub Projects je nový nástroj na správu projektů, aktuálně ve stavu uzavřené beta verze. Mezi hlavní výhody patří integrace s git repozitářem, project management as a code, CLI a API. Cílem práce je si vyzkoušet v tomto prostředí vedení softwarového projektu pomocí agilních metodik vývoje.

Pokyny:
- Proveďte analýzu nástrojů na vedení softwarových projektů. Zaměřte se především na ty, které podporují integraci issues přímo s repozitářem zdrojového kódu.
- Seznamte se způsobem vedení softwarových projektů ve vývojářské firmě, zmapujte jednotlivé postupy a procesy.
- Navrhněte workflow těchto procesů pro GitHub Projects.
- Navržené workflow implementujte.
- Proveďte simulaci virtuálního projektu (či reálný projekt z praxe) v daném workflow.
- Zhodnoťte práci s GitHub Projects oproti stávajícímu řešení v softwarové firmě.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 8. prosince 2021 v Praze.

Bakalářská práce

PROJEKTOVÉ ŘÍZENÍ POMOCÍ GITHUB PROJECTS

Adam Warzel

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Josef Gattermayer, Ph.D.
16. února 2023

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2023 Adam Warzel. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Warzel Adam. *Projektové řízení pomocí GitHub Projects.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Brně dne 16. 2. 2023 .....

Abstrakt

Tato práce se zabývá analýzou a testováním nového nástroje na řízení projektů GitHub Projects. Práce popisuje podrobně možnosti nástroje, porovnává jeho funkce s jinými nástroji a popisuje jeho praktické použití při řízení virtuálního projektu agilní metodikou. V závěru práce je uvedeno porovnání použití GitHub Projects s použitím nástroje Jira Software v reálné softwarové firmě. Tato analýza může čtenáři pomoci s výběrem vhodného nástroje pro řízení projektů.

Klíčová slova GitHub Projects, projektové řízení, softwarové projekty, řízení požadavků, agilní metodiky, Scrum, GitHub Issues

Abstract

The thesis comprises of an analysis and testing of the new project management tool GitHub Projects. The work describes in detail the possibilities of the tool, compares its functions with other tools and describes its practical use in managing a virtual project using agile methodology. At the end of the work is a comparison of using GitHub Projects with using the Jira Software tool in a real software house. This analysis can help the reader to choose a suitable project management tool.

Keywords GitHub Projects, project management, software projects, issue tracking, agile, Scrum, GitHub Issues

Úvod

V dnešní době existuje mnoho nástrojů pro správu vývoje softwarových projektů. Některé z nich jsou rozsáhlé aplikace, obsahující nespočet funkcí, jiné jsou zase jednoduché nástroje, poskytující základní funkcionalitu bez rušivých elementů. Nedávno vyšel nový nástroj integrovaný se známou vývovojou platformou GitHub, který se jmenuje GitHub Projects. Tato práce může pomoci s rozhodnutím, jestli z dostupných nástrojů zvolit právě tento.

Práce se zabývá analýzou nástrojů řízení softwarových projektů a návrhem i implentací pracovních postupů při vedení softwarových projektů v nástroji GitHub Projects.

První, spíše teoretická, část práce se věnuje představení agilního řízení projektů, porovnání běžně používaných nástrojů a analýze nového nástroje GitHub Projects. V další části práce jsou analyzovány procesy a postupy při vedení softwarového projektu ve firmě, poté je navrhnuto a implementováno workflow (pracovní postup) těchto procesů v GitHub Projects. Pomocí této implementace je pak simulován průběh virtuálního projektu.

Cíle

Úkolem práce je analyzovat prostředí GitHub Projects jako nástroj pro řízení softwarového projektu pomocí agilních metodik vývoje. GitHub Projects je nový nástroj pro správu softwarových projektů vydaný v roce 2022. Mezi výhody nástroje patří integrace s git repozitářem, project management as a code, CLI a API.

Cílem první části práce je představit agilní řízení projektů a srovnat existující nástroje na vedení softwarových projektů, se zaměření je zejména na ty, které podporují integraci s repozitářem zdrojového kódu.

Úkolem druhé části je analyzovat způsob vedení softwarových projektů ve vývojářské firmě, navrhnout pracovní postupy těchto procesů pro GitHub Projects a implementace postupů v tomto prostředí. Dalším cílem je simulovat virtuální projekt za použití tohoto workflow.

Záměrem poslední části je zhodnocení práce s GitHub Projects oproti stávajícím řešením v softwarové firmě. Výsledkem práce je analýza, která může čtenáři pomoci s výběrem vhodného nástroje pro projektové řízení softwarového projektu.

Seznam zkratek

API	Application Programming Interface, aplikační programovací rozhraní
CI/CD	Continuous Integration, Continuous Deployment, kontinuální sestavení a nasazení
CLI	Command Line Interface, rozhraní příkazové řádky
GUI	Graphical User Interface, grafické uživatelské rozhraní
HTTP	Hypertext Transfer Protocol, síťový protokol aplikační vrstvy
YAML	Yet Another Markup Language, druh značkovacího jazyka
REST	Representational State Transfer

Kapitola 1

Agilní řízení softwarových projektů

Agilní metodiky jsou v dnešní době dominantní způsob řízení softwarových projektů. V průzkumu firmy digital.io z roku 2021 odpovědělo 94 % dotazovaných firem, že uplatňuje agilní postupy a principy. Ze zkoumaného vzorku 86 % firem řeklo, že agilní postupy využívá pro řízení vývoje software (mezi ostatní oblasti uplatnění agilních postupů patří například IT oddělení, marketing, vývoj hardware). [1]

Slovo „agilní“ v tomto kontextu znamená schopnost provádět změny a zároveň na ně reagovat. Znamená schopnost adaptovat se na změny, které jsou v prostředí softwarového vývoje velmi časté. Agilní vývoj není jedna konkrétní, dobře definová metodika, ale jde o zastřešující termín pro množinu postupů a metodik, které jsou založeny na principech vyjádřených v takzvaném Manifestu Agilního vývoje software. [2]

1.1 Principy agilního řízení projektů

Hlavními aspekty agilních metodik jsou jednoduchost a rychlost. Při vývoji se tým věnuje funkcionalitě potřebné a požadované v daném okamžiku. Funkcionalita je rychle vyvinuta a předána zákazníkovi, aby potom mohla být od zákazníka předána zpětná vazba a podle ní upravena specifikace požadavků. Vývojová metodika je zpravidla označována za agilní, když je vývoj inkrementální (produkt je dodáván iterativně, po menších částech), vývojáři a zákazník aktivně komunikují během vývoje, a když je metodika adaptivní, tedy schopná pokrýt změny v průběhu vývoje. [3]

1.1.1 Agilní manifest

Centrální myšlenky agilního vývoje jsou vyjádřeny v Agilním manifestu. Ten byl sepsán v roce 2001 skupinou odborníků z oblasti softwarového inženýrství. Myšlenky agilního vývoje ale vznikaly už před sepsáním tohoto manifestu v devadesátých letech.

Manifest agilního vývoje software

Objevujeme lepší způsoby vývoje software tím, že jej tvoříme a pomáháme při jeho tvorbě ostatním. Při této práci jsme dospěli k těmto hodnotám:

- **Jednotlivci a interakce** před procesy a nástroji
- **Fungující software** před vyčerpávající dokumentací
- **Spolupráce se zákazníkem** před vyjednáváním o smlouvě
- **Reagování na změny** před dodržováním plánu

Jakkoliv jsou body napravo hodnotné, bodů nalevo si ceníme více.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

© 2001, výše zmínění autoři
Toto prohlášení může být volně kopírováno v jakékoli formě, ale pouze v plném rozsahu včetně této poznámky. ([4])

Za tímto manifestem stojí 12 principů agilního vývoje softwaru. Toto jsou už konkrétnější poučky, kterými by se měly řídit týmy implementující agilní přístup k řízení softwarových projektů.

”

1. Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
2. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
3. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
4. Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
5. Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
6. Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
7. Hlavním měřítkem pokroku je fungující software.
8. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.
9. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.
10. Jednoduchost–umění maximalizovat množství nevykonané práce–je klíčová.
11. Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
12. Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

“ ([4])

1.2 Agilní metodiky

Metodik pro řízení softwarových projektů založených na agilních principech vzniklo od vydání Agilního manifestu mnoho. Některé z nich jsou velmi populární, jiné se dnes již téměř nepoužívají. Zde je přiblíženo jen několik vybraných z nich. Nejdetailněji je popsána metodika Scrum, jejíž použití ve zkoumané firmě je analyzováno dále v práci (4.1.1), a která je tedy aplikována pro řízení virtuálního projektu s pomocí nástroje GitHub Projects.

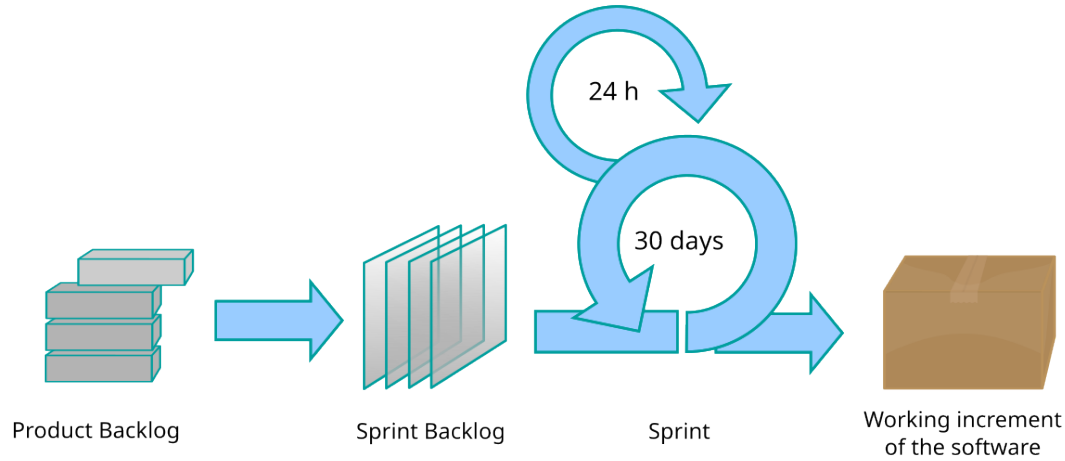
1.2.1 Scrum

Jedná se o dnes asi nejrozšířenější agilní metodiku. Podle průzkumu společnosti KMPG z roku 2019 je Scrum používán v 78 % dotazovaných firem. [5] V průzkumu digital.ai z roku 2021 odpovědělo 66 % dotázaných firem, že používá přímo metodiku Scrum a dalších 15 % odpovědělo, že používá metodiku od Scrumu odvozenou. [1]

Scrum je agilní framework, pracovní rámec. Popisuje jak pracovat v týmu na softwarových projektech. Definuje role lidí, kteří se vývoje účastní, a popisuje průběh vývojového cyklu. [6] Tím se liší od dále popsaných metodik Kanban a Extrémní programování, které se spíše soustředí na správné praktiky a nedefinují tolik podrobně, jak má vypadat tým lidí účastnící se vývoje a vývojový cyklus.

Jádrem Scrumu je iterativní proces postupného, inkrementálního, přidávání funkcionality. Vývojový cyklus probíhá v iteracích, ve Scrumu nazývaných Sprinty. [6] Během Sprintu se členové týmu jednou denně setkávají kvůli vzájemné kontrole aktivit a pokroku, aby mohl tým reagovat na případné změny a problémy. Tým se během Sprintu snaží splnit cíl Sprintu, který je stanoven podle požadavků na produkt. Výstupem Sprintu je inkrementace funkcionality. Popisovaný proces, tzv. kostra Scrumu, je znázorněn na obrázku 1.1. Pojmy produktový backlog a backlog Sprintu jsou vysvětleny níže v podsekci 1.2.1.2. [7, s. 25–26]

Obrázek 1.1 Kostra Scrumu [8]



1.2.1.1 Scrumový tým

Lidé pracující na projektu řízeném metodou Scrumu jsou organizováni do týmů. Tým je sestaven tak, aby byl v práci na projektu soběstačný. Pro práci na projektu tedy neexistuje závislost na

nikom mimo tým. Je samoorganizovaný, rozhodování o tom kdo a kdy bude na čem pracovat probíhá interně v týmu. Tým má až 3–10 členů. Menší týmy jsou efektivnější v komunikaci a mají lepší produktivitu. Pokud je tým příliš velký, je dobré zvážit reorganizaci ho více spolupracujících týmů, které budou sdílet Product Ownera a pracovat na stejném produktu. Členové týmu jsou rozděleni do následujících rolí:

- **Vývojáři** jsou zodpovědní za přidávání funkcionality ve Sprintech. Konkrétní dovednosti se liší podle domény projektu. Jedná se o programátory, testery, databázisty atd.
- **Product Owner** je zodpovědný za definici a prioritizaci požadavků na projekt, stará se o produktový backlog. Je to člověk, který zastupuje zájmy všech zainteresovaných subjektů, takzvaných stakeholderů.
- **Scrum Master** je zodpovědný za implementaci Scrumu. Dohlíží používání praktik Scrumu, pomáhá týmu v samoorganizaci a pomáhá Product Ownerovi s definicí produktového cíle a organizací požadavků.

[6]

1.2.1.2 Artefakty Scrumu

Artefakty ve Scrumu představují práci nebo hodnotu. Jejich smyslem je zvýšení transparentnosti důležitých informací. Umožnují měření pokroku práce na projektu. [6]

Product Backlog. Backlog je anglické slovo, které znamená seznam nedodělané práce. Backlog produktu v metodice Scrum, je seznam veškeré práce na projektu. Tento seznam je seřazen podle priority položek. [6] Položky seznamu kromě popisu obvykle obsahují i odhad komplexity a prioritu. Prioritní úkoly jsou zpracovány detailněji, méně prioritní mohou zůstat popsány více obecně a budou zpracovány až jejich priorita v průběhu projektu naroste.

V Backlogu produktu se také nachází Produktový Cíl, popisující budoucí stav produktu, ke kterému se scrumový tým plánuje propracovat. Ostatní položky definují ”co”naplní tento Produktový cíl. Je to dlouhodobý cíl pro tým. Než je stanoven nový Produktový cíl, musí ten starý být naplněn, nebo opuštěn.

Požadavky do Backlogu produktu přidává Product Owner. Po celou dobu vývoje probíhá tzv. Refinement Backlogu produktu, což je proces, kdy celý tým diskutuje položky backlogu, přidává k nim detaily detaily, odhady a rozděluje položky na menší úkoly. [6]

Sprint Backlog. Backlog Sprintu. Skládá ze stanoveného cíle Sprintu, vybraných položek z Backlogu produktu a plánu jak cíl splnit a tím doručit inkrement. Je to plán tvořen vývojáři, protože právě oni budou pracovat na jeho splnění. Stav položek je aktualizován v průběhu Sprintu. Tým tak může kontrolovat průběh Sprintu během každodenních schůzek (skrumáže).

Pokus se během práce na Sprintu ukáže, že velikost potřebné práce je jiná, než tým očekával, tým může s Product Ownerem vyjednat změnu rozsahu Sprintu. [6]

Inkrement. Konkrétní funkční změna, krok za dosažením produktového cíle. Během jednoho Sprintu je možné vytvořit více inkrementů.

Definition of Done je název formálního popisu stavu hotového inkrementu. Jakmile se položka backlogu dostane do stavu splňujícího *Definition of Done*, vzniká inkrement. [6]

1.2.1.3 Události Scrumu

Události Scrumu jsou všechny časově ohraničené události ve Scrumu. Každá událost definovaná Scrumem je formální příležitost zkontrolovat a upravit artefakty Scrumu. Události Scrumu poskytují týmu pravidelně zpětnou vazbu, tým tak může včas reagovat a upravovat další plán vývoje. Události Scrumu také minimalizují potřebu konat jiné meetingy, které nejsou definovány ve Scrumu. [6]

Sprint. Sprint je vývojová iterace Scrumu. Sprinty mají konstantní délku v řádu týdnů, ne více než jeden měsíc. Mezi Sprinty není žádná časová mezera, nový Sprint začíná okamžitě po konmci předchozího. Veškeré vývojové práce na produktu i ostatní Scrumové události probíhají v rámci Sprintu.

Během Sprintu platí že:

- Nejsou provedeny žádné změny, které by ohrozili dosažení cíle Sprintu
- Kvalita produktu neklesá
- Produktový backlog je upravován dle potřeby
- Rozsah Sprintu je ujasňován s Product Ownerem a je možné vyjednávat o změně rozsahu, pokud tým během práce ve Sprintu přichází na nové skutečnosti

Sprint může být zrušen, pokud se Cíl Sprintu stane neaktuálním a zastaralým. Zrušit Sprint může pouze Product Owner. [6]

Plánování Sprintu. Naplánováním Sprintu iterace začíná. Na plánování spolupracuje celý tým a v případě potřeby je možné přizvat i lidi, kteří nejsou členy týmu. Výstupem tohoto meetingu je Sprint Backlog. Tým tedy stanoví Cíl Sprintu, které položky Produktového backlogu budou vypracovány a jakým způsobem budou vývojáři pracovat na jednotlivých položkách. [6]

Při stanovování rozsahu Sprintu je vhodné brát ohled na objem dokončené práce v předchozích Sprintech. Dobře fungující tým má stabilní rychlost, dokočuje obdobný objem práce každý Sprint. Při rozhodování o rozsahu plánovaného Sprintu by se měl tým tedy držet obdobného rozahu, který se týmu podařilo dokončit v proběhlých Sprintech. [9]

Daily Scrum. V doslovném překladu denní skrumáž, tedy shluk, tlačenice. Je to meeting, podle kterého je celý framework pojmenovaný. Jedná se o každodenní patnáctiminutové setkání vývojářů. Mohou se ho účastnit i Product Owner a Scrum Master, pokud pracují na položkách Sprint Backlogu, ale primárně je to meeting pro vývojáře. Účelem meetingu je zkontrolovat postup v plnění Cíle Sprintu a případně upravit plán dalšího postupu. Přesná podoba meetingu záleží na vývojářích. Výstupem meetingu je plán na další den vývoje. [6]

Sprint Review. Během Sprint Review tým prezentuje výsledky Sprintu zainteresovným subjektům a diskutuje pokrok v dosažení Produktového cíle. Zainteresované subjekty hodnotí prezentované výsledky a komunikují případné změny požadavků. Podle této zpětné vazby tým plánuje další práci a v případě potřeby upravuje Backlog prduktu. Jde o předposlední událost Sprintu. [6]

Sprint Retrospective. Poslední událost Sprintu. Scrum tým na ní hodnotí jak probíhal Sprint co se týče procesů, nástrojů, interakcí i jednotlivých členů týmu. Tým diskutuje co šlo dobře a na jaké probémy tým narazil v průběhu Sprintu. Výstupen meetingu je nalezení změn, které pomohou zlepšit efektivitu. Tyto změny by měly být zapracovány co nejdříve, mohou být i přidány přímo do Sprint Backlogu následujícího Sprintu. Tento meeting uzavírá Sprint. [6]

1.2.2 Kanban

Metodika Kanban byla původně vytvořena pro použití v průmyslové výrobě v automobilové továrně Toyota Motor Corporation japonským inženýrem jménem Taiichi Ohno. Význam slova Kanban v japonštině je cedule, nebo vývěsní štít. Přestože se nejedná původně o metodiku pro vývoj softwaru, jsou i pro vývoj softwaru důležité cíle jako plynulost a rychlost dodávání produktu, a tak je metodika používána i v tomto odvětví. [10]

1.2.2.1 Kanban principy

Vizualizace práce. Klíčový princip metodiky Kanban, podle kterého se metodika jmenuje. Vizualizace usnadňuje sledování průběhu práce, umožňuje vždy snadno zjistit jaký je aktuální stav. Hlavním nástrojem pro vizualizaci v této metodice jsou tzv. Kanban nástěnky. Na těchto nástěnkách jsou umístěny kartičky reprezentující úkoly, které je třeba splnit. Kartičky s úkoly jsou seskupeny podle stavu. V průběhu práce na položkách se jejich pozice aktualizuje. Nástěnky mohou být jak fyzické, tak virtuální, poskytované softwarovými nástroji pro řízení projektů. Vhodnými nástroji pro vizualizaci jsou také různé grafy, znázorňující počty úkolů ve frontě, rozpracovaných a dokončených úkolů. [10]

Pull model. Zainteresované strany, které výtváří požadavky a zadávají práci, nepřiřazují práci s termíny vývojovému týmu. Místo toho přidávají úkoly do seznamu, backlogu, ze kterého s tým úkoly postupně přebírá, jakmile na to má kapacitu. To umožňuje týmu udržovat kvalitu a nepřekračovat nastavené limity počtu rozpracovaných úkolů [10]

Limitace množství rozpracovaných úkolů. Příliš mnoho rozpracovaných úkolů může vést k častém přepínání kontextu, což je časově nákladné a snižuje efektivitu. Aby se tomu předešlo, tým si zavede limit v počtu rozpracovaných položek. Práce na nových položkách ze seznamu úkolů, tedy backlogu, může být týmem zahájena jen pokud by tím nedošlo k překročení tohoto limitu. [10]

Měření výkonu a zlepšování se Aby se tým mohl zlepšovat, musí měřit svou efektivitu a výkon. Měření průměrného času, který trvá úkolu projít celým cyklem vývoje, umožňuje zkoušet zavádění nových postupů, monitorovat jejich vliv na výkon a podle toho se rozhodnout, které postupy dále použít. [9][10]

1.2.2.2 Porovnání s metodikou Scrum

Velký rozdíl mezi metodikou Kanban a metodikou Scrum je ten, že v metodice Kanban není vývoj rozdělen na iterace. Probíhá kontinuálně, doručení nových funkcionalit a změny v pracovním postupu mohou nastat kdykoliv, ne jen na konci iterace. Kanban také tým nerozděluje do rolí. [11]

Metodika Kanban je volnější, více flexibilní než metodika Scrum. Kromě několika základních principů nestanovuje konkrétní postupy, které vývojový tým má dodržovat. Neříká jaké meetingy se mají konat. To vše si rozhoduje tým, implementující tuto metodiku.

Metodika je zejména vhodná pro řízení práce týmů, jejichž práci nelze jednoduše rozdělit na iterace. Například týmy jejichž kolem je údržba a podpora. [9]

1.2.3 Extrémní programování

Extrémní programování je metodika, jejíž hlavní cíle jsou produkce kvalitního softwaru a vyšší kvalita života vývojového týmu. Tato metodika definuje hodnoty, důležité k dosažení těchto cílů, a konkrétní praktiky, jejichž použití povede k naplnění těchto hodnot.

Extrémní programování je vhodné na projektech kde

- se dynamicky mění požadavky,
- je potřeba mitigovat rizika způsobená termíny a novými technologiemi,
- pracuje malý, kolokovaný tým[1],
- technologie umožňuje psaní automatických jednotkových, integračních a systémových testů.

[12]

1.2.3.1 Hodnoty Extrémního programování

Jednoduchost. Tým pracuje pouze na těch věcech, které jsou v daný moment potřeba. Nedělá nic navíc, nesnaží se předpovědět, co by se mohlo stát a jaké další požadavky by mohl zákazník v budoucnu mít. Cílem je vždy dodat co nejmenší a nejjednodušší změnu, která přináší zákazníkovi hodnotu. [9]

Komunikace. Všichni členové týmu spolu každý den osobně mluví, spolupracují na návrhu řešení, kódu i testování. Spolupracují na řešení problémů, které je blokují. [9]

Zpětná vazba. Tým pracuje v krátkých iteracích, s každou iterací je dodán funkční software. Nejen na konci iterace, ale i v jejím průběhu ukazuje tým implementované funkcionality tak, aby získal co nejkvalitnější a nejčastější zpětnou vazbu a mohl na ni hned reagovat. Zodpovědnost za proces leží na týmu. Tým musí upravovat své procesy tak, aby vyhovovaly produktu, nikde ne naopak. [9]

Respekt. Aby vývojový tým dobře fungoval, je potřeba vzájemný respekt a vnímání hodnoty jednotlivých členů navzájem. Tým musí respektovat přání a potřeby zákazníka a stejně tak zákazník technické znalosti a dovednosti týmu. Management musí respektovat tým a jeho zodpovědnost za organizaci práce v týmu. [9]

Odvaha. Metodika Extrémní programování vyžaduje od členů týmu odvahu. Odvahu nazývat věci pravými jmény, vypořádat se s problémy a nezametat je pod koberec, přijmou zodpovědnot za špatné odhady, udělat změny, když je to potřeba. [9]

1.2.3.2 Praktiky

Extrémního programování obsahuje množinu praktik pro softwarový vývoj, které jsou navzájem propojené, dohromady popisují vývojový proces a tvoří tak jádro metodiky.

Sezení společně. Tým sedí společně v kanceláři, aby mohl kdykoliv efektivně komunikovat bez jakýchkoliv překážek. [12]

[1] Pracující na jednom místě, nikoliv distribuovaný po různých městech, či státech

Kompletní tým. Tým je tvořen multifunkční skupinou lidí, obsahující veškeré know-how a veškeré role, potřebné pro práci na projektu. Díky tomu nevznikají závislosti na jiných týmech a lidech mimo tým. Lidé, kteří po sobě navzájem něco potřebují, se denně setkávají v rámci týmu. [12]

Informativní pracovní prostředí. Pracovní prostředí by mělo být uspořádáno tak, aby umožňovalo efektivní komunikaci z očí do očí, ale v případě potřeby i poskytovalo soukromí členům týmu, když to potřebují. Práce týmu musí být transparentní jak pro členy týmu, tak pro všechny zainteresované skupiny mimo tým. [12]

Energická práce. Členové týmu jsou nejvíce efektivní ve vývoji softwaru, když se mohou soustředit a nejsou rozptylováni. Tato praktika říká, že tým se musí starat, aby členové byly ve fyzické i mentální pohodě, aby se nepřepracovávali a aby respektovali potřeby svých kolegů. [12]

Párové programování. Veškerý produkční kód by měl být napsán pomocí párové programování. Párové programování je, když 2 lidé sedí u jednoho počítače a spolupracují na psaní kódu. Výhodou tohoto postupu je simultánní code review, kontrola kódu, a efektivnější řešení problémů u kterých by se jeden člověk mohl zaseknout. [12]

User Stories. Produkt by měl být popsaný tak, jak k němu budou přistupovat jeho uživatelé. Tento popis by měl být rozdělený na malé části zvané User Stories, příběhy. Každé User Story popisuje nějaký druh interakce uživatele s produktem. Tyto user stories pak slouží i k plánování vývoje. [12]

Týdenní cyklus. Týdenní cyklus je základní iterace v metodice. Tým se sejde na začátku týdne, aby zhodnotil postup doažený v minulé iteraci, zákazník vybere, které user stories by chtěl mít doručeny příští týden a tým rozhodne o postupu doručení těchto úkolů. Cílem iterace je mít na konci týdne vyvinuté a otestované funkce, které realizují vybrané user stories. Důvodem ustanovení časově omezených iterací je, že tým tak bude mít pravidelně příležitost prezentovat svou práci a získávat zpětnou vazbu od zákazníka. [12]

Čtvrtroční cyklus. Vydání nové verze produktu se děje jednou za čtvrt roku. Zákazník představí týmu plán funcí, které chce mít hotové v daném kvartálu a tým podle toho může sestavit plán, jak tento cíl splnit. Protože se jedná o relativně vysokoúrovňový plán na delší časové období, může se snadno stát, že tým zjistí, že pořadí user stories, nebo jejich počet v daném čtvrtročním cyklu je potřeba v průběhu cyklu měnit. Je vhodné aby tým na konci každého týdenního cyklu zkontroloval čtvrtoční plán. V případě potřeby aktualizace je tak možné rychle informovat zákazníka. Účelem čtvrtočního cyklu je udržet přehled nad celkovým plánem projektu a zasazení jednotlivých user stories a týdenních cyklů do tohoto dlouhodobého plánu. [12]

Rezerva. Do kvartálního i týdenního cyklu je vhodné zařadit i úkoly s nižší prioritou, které pak mohou být v případě potřeby ze sprintu vypuštěny. Tým musí počítat s inherentní nepřesností odhadů. Tato rezerva týmu pomůže získat dobrou šanci pro splnění plánů. [12]

Desetiminuté sestavení. Celý vyvýjený systém by měl být možné automaticky sestavit a otestovat do deseti minut. Čím déle sestavení a otestování trvá, tím delší je typicky doba mezi jednotlivými otestováními a tím déle trvá nalezení chyb. [12]

Průběžná integrace. Při přidání nových změn do kódu je vhodné, aby se provedlo sestavení a otestování celého systému. To umožnuje rychlé zachycení a opravení možných integračních chyb. Dodržení praktiky desetiminutového sestavení, uvedené výše, zaručuje časovou efektivitu této praktiky. [12]

Nejdříve testy. Namísto rozšířeného postupu vývoj kódu, vývoj testu, spuštění testu je lepší nejdříve napsat test a až poté kód produktu, který zaručí splnění testu. Tento postup zaručuje rychlejěší zpětnou vazbu pro vývojáře kódu, který tak může rychleji identifikovat a vyřešit možné problémy. [12]

Inkrementální design. Tým by měl vždy před tím, než se pustí do implementace funkcí, věnovat čas porozumění designu systému na vyšší úrovni, a potom se věnovat konkrétnějším aspektům tohoto designu, když pracuje na konkrétních úkolech. Díky tomu bude možné dělat designová rozhodutí vždy když je potřeba, na základě aktuálních informací.

Extrémní programování je stejně jako metodika Scrum iterativní, definuje délku iterací, frekvenci vydání nových verzí, a říká jak má vypadat celý vývojový cyklus. Obě metodiky podle principů agilního vývoje kladou důraz pravidelnou zpětnou vazbu a přizpůsobení se aktuálním okolnostem. Metodika Extrémního programování ale narozdíl od metodiky Scrum nedefinuje konkrétní role v týmu, vyžaduje, aby tým byl kolokovaný a popisuje konkrétní techniky vývoje softwaru, které má vývojový tým používat.

Kapitola 2

Nástroje pro řízení softwarových projektů

V této kapitole jsou popsány vybrané nástroje pro řízení sotfwarových projektů, které podporují integraci s repozitářem zdrojového kódu.

GitHub Projects a **GitLab nástroje** jsou přímou součástí webových aplikací pro správu repozitářů zdrojového kódu. Tento druh nástrojů nabízí jednoduchost nastavení a automatickou integraci s repozitářem zdrojového kódu.

Dále jsou popsány samostatné nástroje, a to open source nástroj **Redmine** a komerční nástroj **Jira Software** od společnosti Atlassian.

2.1 Jira Software

Jira je název rodiny softwarových nástrojů pro řízení projektů a lidí od firmy Atlassian. Do této skupiny patří nástroje Jira Work Management, nástroj určený pro řízení marketingových, HR, finančních a podobných oddělení, Jira Service Management, nástroj pro správu požadavků o služby, tiketů, určený pro IT oddělení, či jiná oddělenení, poskytující podporu a služby a konečně nástroj Jira Software, určený k řízení softwarových projektů. [13] Nástroj Jira Software byl poprvé vydán v roce 2001 pod názvem Jira 1.0. [14]

Podle výroční zprávy State of Agile firmy digital.io z roku 2022 je Jira Software s 66 % nejpoužívanějším nástrojem pro agilní řízení projektů, i když tržní podíl klesl ze 72 % v roce 2021. [1] Jira Software je platforma, která poskytuje správu úkolů, issues, a funkce pro agilní řízení. Nástroj je webová aplikace. Může být používán z cloudu firmy Atlassian, nebo může být spuštěn na vlastní infrastruktuře. Nástroj je designován jako vysoce upravitelný a přizpůsobitelný, jak vizuálně tak funkčně. [13]

Cena nástroje Jira Software

Nástroj Jira Software je v cloudové verzi zdarma pro projekty, které mají do 10 členů. Verze pro více uživatelů nebo verze pro spustění na vlastní infrastruktuře je nutné měsíčně, nebo ročně předplácet. Existuje několik předplatitelských plánů podle požadovaného počtu uživatelů a pokročilých funcí, které budou pro nástroj zapnuty. Cena začíná na 77,5 $ za měsíc. Verze nástroje, která je zadarmo neobsahuje některé funkce jako je třeba plánování kapacit a archivování projektů. [13]

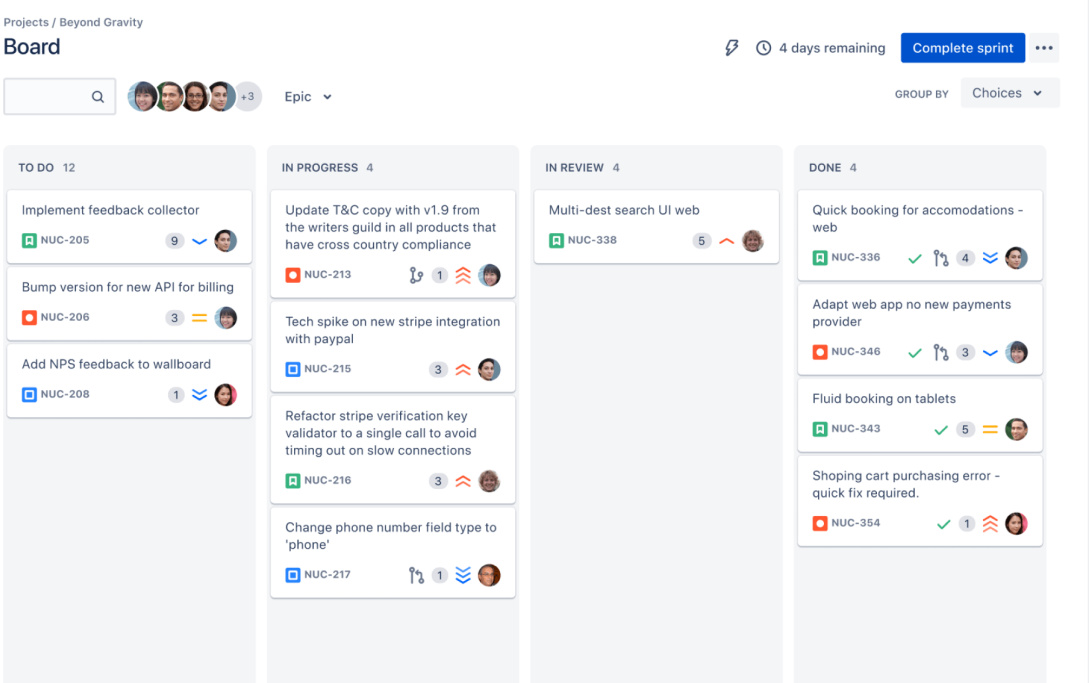
Funkce nástroje Jira Software

Mezi funkce, které Jira Software nabízí patří:

- Projektové nástěnky
- Backlog
- Vyhledávání úkolů pomocí vestavěného dotazovacího jazyka JQL
- Nastavitelný životní cyklus úkolů
- Dlouhodobé plánování pomocí funkce Roadmap
- Automatizaci pomocí vestavěného GUI nástroje
- REST API a webhooks
- Integrace s více než 3000 komukačními, verzovacími, CI/CD a dalšími nástroji
- Atlassian Marketplace obsahující applikace a rozšiřující zásuvné moduly[1]

[13]

Obrázek 2.1 Nástěnka sprintu v nástroji Jira Software [15]



[1] Některé aplikace a rozšiřující moduly jsou placené

2.2 Redmine

Redmine je open source nástroj pro řízení projektů. Zdrojový kód je poskytován pod licensí GNU General Public License v2. Nástroj je k dispozici k použití zdarma pro jakýkoliv tým a organizaci. Stejně jako u ostatních nástrojů v této práci se jedná o webovou aplikaci. Tato aplikace je implementovaná pomocí frameworku Ruby on Rails. [16]

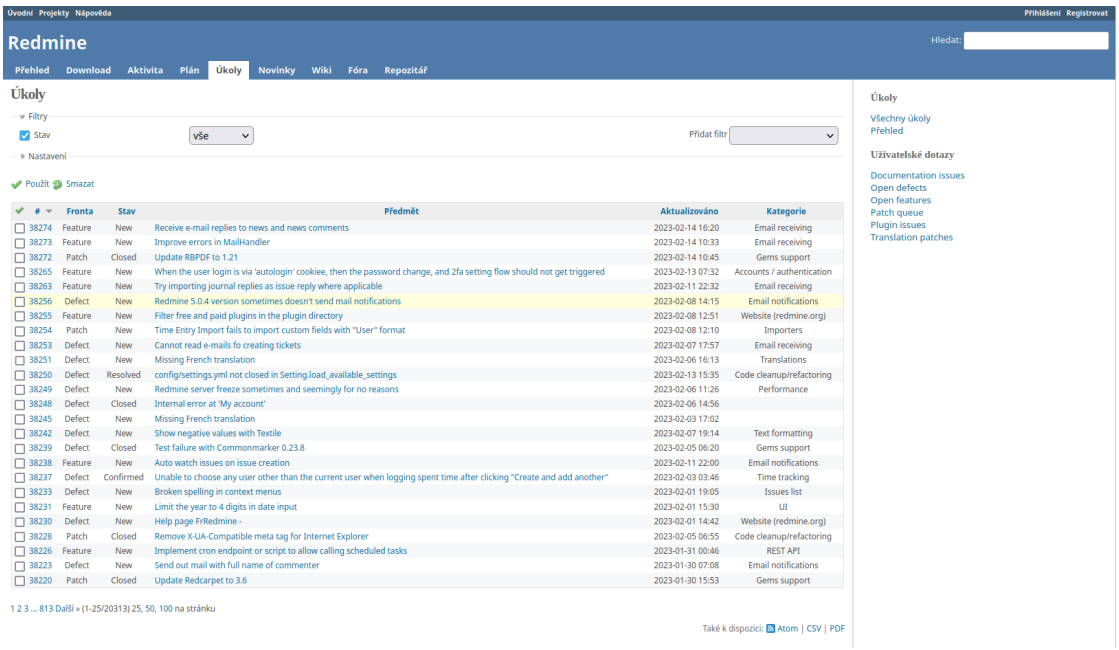
Redmine je možné provozovat na vlastní infrastruktuře. Pro provoz aplikace je potřeba server s interpretem jazyka Ruby a jeden z podporovaných databázových serverů. Podporovány jsou databázové systémy MySQL, PostgreSQL, Microsoft SQL Server a SQLite 3. Druhou možností je využít dedikovaný Redmine hosting poskytovaný třetí stranou. [17]

Mezi fukce které nástroj nabízí patří:

- Správa úkolů, issues
- Možnost přidání vlastních datových polí do úkolů
- Měření času stráveného na úkolu
- Douhodobé plánování pomocí funkce Roadmap
- Správa souborů a dokumentů
- Integrace s repozitářem zdrojového kódu
- REST API
- Projektová wiki
- Projektové forum
- Zásuvné moduly

[16]

Obrázek 2.2 Seznam úkolů v nástroji Redmine s českou lokalizací [18]



2.3 GitLab

GitLab je DevSecOps platforma od stejnojmenné společnosti. Nabízí služby hostování Git repozitářů, CI/CD, řízení projektů a další. Platforma je webová aplikace, která může být nasazena ve vlastním prostředí, nebo na cloudové infrastruktuře společnosti Gitlab [19]. GitLab umožňuje uživatelům zakládat projekty, což jsou v jádru Git repozitáře zdrojového kódu, a k těmto projektům přidávat úkoly, úlohy, plány, dokumentaci. Funkce pro řízení projektů zde nemají narozdíl od Projects na platformě GitHub své vlastní jméno. [20]

Cena nástrojů GitLab pro řízení projektů

Nástroje pro řízení projektu jsou ke každému projektu zdarma. Projekt, repozitář kódu, lze založit na platformě, běžící na gitlab.com zdarma. Nebo je možné nasadit GitLab Community Edition na vlastní infrastruktuře. Community Edition je open source verze platformy, která je volně k dispozici. [19]

Funkce

Mezi funkce, které GitLab nástroje nabízí patří:

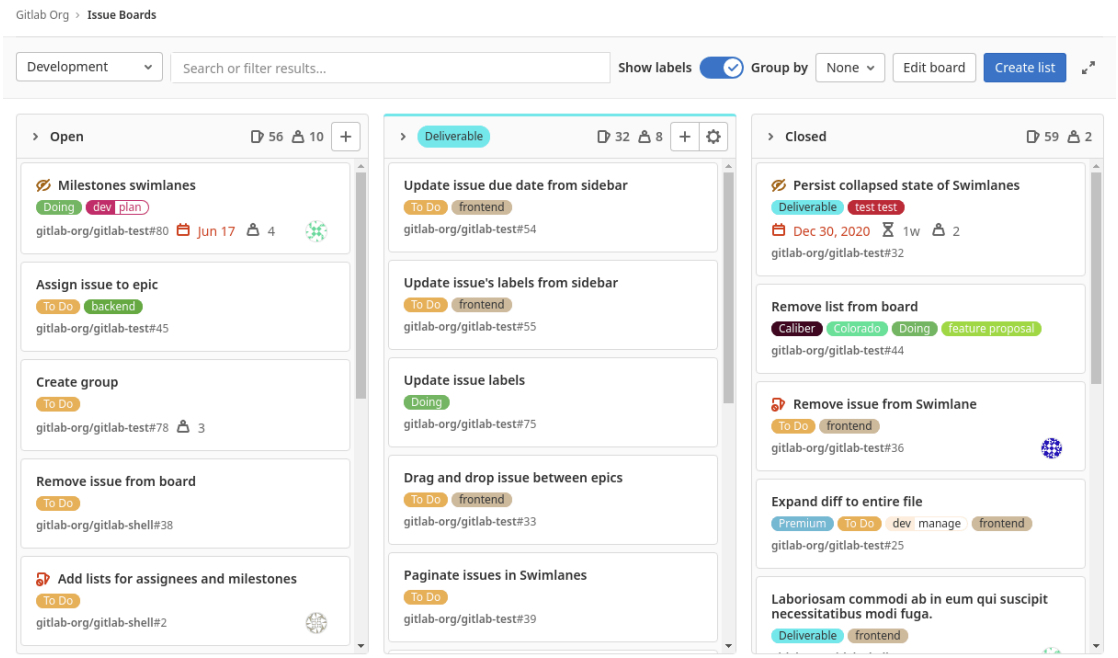
- Úkoly a podúkoly, v GitLabu zvané issues a tasks
- Milníky
- Nástěnky úkolů
- Sledování času stráveného na úkolu
- Integrace s Git repozitářem
- REST API
- GraphQL API
- Projektová wiki

Funkce nabízené pouze v placených verzích:

- Úkoly typu Epic, pro sledování větších úkolů přes více repozitářů
- Dlouhodobé plánování pomocí funkce Roadmap

[20]

Obrázek 2.3 Nástěnka s úkoly v nástroji GitLab [21]



2.4 GitHub Projects

GitHub Projects je nastavitelný a přizpůsobitelný nástroj na plánování projektů na platformě GitHub. GitHub je webový poskytovatel hostingu pro repozitáře verzovacího systému Git, který kromě hostování repozitářů nabízí i nástroje pro podporu vývoje softwaru. Mezi tyto nástroje patří například GitHub Codespaces, nástroj umožnující vytvoření vývojového prostředí v cloudu, GitHub Actions, nástroj pro CI/CD, a v neposlední řadě GitHub Issues, nástroj pro správu nápadů, úkolů a chyb v softwaru. Právě pod GitHub Issues patří i nástroj GitHub Projects. [22]

Repozitář a nástroje jsou standardně umístěny v cloudu GitHubu, alternativně je lze nasadit ve vlastním prostředí, při použití služby GitHub Enterprise Server. [23]

Nástroj Issues umožnuje zaznamenávat k repozitářům issues, což jsou záznamy pro jednotlivé úkoly a chyby. Nástroj Projects dává uživateli možnost tyto záznamy záznamy spravovat a pracovat s nimi různými způsoby. Projects umožňuje vytvářet různé pohledy a přehledy nad těmito záznamy, přiřazovat záznamy k různým projektům, přidávat jim atributy, měnit jejich stavy a další možnosti. [24]

GitHub Projects existuje ve dvou verzích. Projects (classic) je původní verze, na kterou lze narazit na stávajících projektech, ale v době psaní této práce (rok 2022) již nelze v aktuální verzi GitHubu zakládat nové projekty v Projects (classic). Projects bez přídomku (classic) je aktuální verze, která byla vydána roku 2022. Tato práce se dále bude zabývat pouze aktuální verzí GitHub Projects. [24]

Projects je v jádru nastavitelný tabulkový procesor, připomínající známe tabulkové procesory jako je Microsoft Excel nebo Google Sheets, který je integrován s Git repozitářem a ekosystémem GitHubu. Poskytuje integraci pro pull requests (žádosti o revizi změn v kódu) i pro automatické procesy implementované pomocí nástroje GitHub Actions. [24]

GitHub Projects poskytuje automatickou integraci s repozitářem kódu na GitHubu. Nástroj je součástí ekosystému GitHubu a součástí webové aplikace GitHub, takže neposkytuje integraci s repozitáři který není umístěn na GitHubu. [24]

Cena nástroje GitHub Projects

GitHub Issues a tedy i GitHub Projects jsou k dispozici zdarma ke všem všem repozitářům na platformě GitHub. Repozitář lze založit na platformě zdarma. [25]

Funkce nástroje GitHub Projects

Mezi funkce, které tento nástroj nabízí patří:

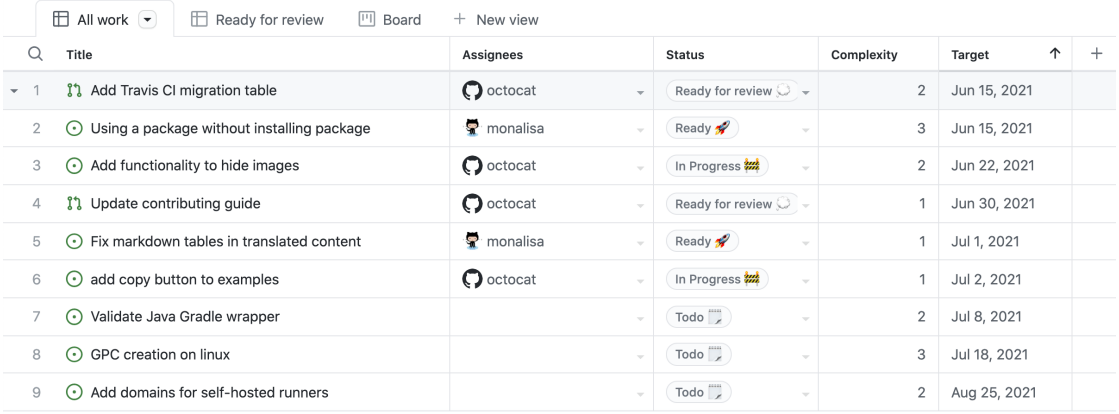
- Správa projektů a úkolů, issues
- Škála nastavitelných pohledů na úkoly
- GraphQL API
- Automatizace
- Nastavení přístupu k projektům podle uživatele a organizace
- Vytváření grafů nad projekty
- Integrace s Git repozitářem

Funkce nacházející se ve stavu uzavřené beta verze, které zatím veřejně dostupné:

- *Tasklists*, tedy seznamy úkolů a hierarchie úkolů
- Pohled typu *Roadmap*, dlouhodobý časový plán ukazující milníky projektu

[24]

Obrázek 2.4 GitHub Projects ilustrační tabulkový přehled [26]



2.5 Porovnání funkcí nástrojů

V tabulce 2.1 je zobrazeno porovnání nástrojů pro řízení projektů podle některých vybraných funkcí.

Tabulka 2.1 Porovnání nástrojů pro řízení projektů [27][16][20][24]

Funkce	Jira Software	Redmine	GitLab nástroje	GitHub Projects
Správa úkolů	Ano	Ano	Ano	Ano
Roadmap	Ano	Ano	V placené verzi	Beta verze
Sprint Board	Ano	Jako zásuvný modul	Ano	Ano
Přidání vlastních atributů úkolům	Ano	Ne	Ne	Ano
Grafy	Ano	Jako zásuvný modul	V placené verzi	Ano
Integrace s repozitářem kódu	Jako zásuvný modul	Pouze prohlížení	Jen pro GitLab	Jen pro GitHub
Automatizace pomocí GUI	Ano	Placený zásuvný modul	Ne	Pouze základní
REST API	Ano	Jako zásuvný modul	Ano	Ne
GraphQL API	Ne	Ne	Ano	Ano
Webhook	Ano	Jako zásuvný modul	Ano	Ano

Kapitola 3
Analýza GitHub Projects

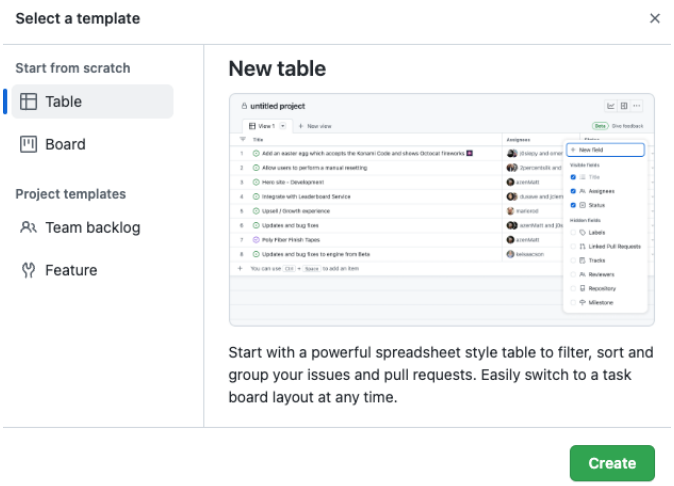
V této kapitole jsou podrobněji rozebrány možnosti, funkce a nastavení, které GitHub Projects nabízí.

3.1 Projekty

Projekty Github Projects mohou být vytvořeny buď jako projekty organizace, nebo projekty uživatele. Projekty organizace jsou pro sledování úkolů a pull requestů týkajících se repozitářů organizace, projekty uživatele lze použít pouse s repozitáři uživatele samotného.

Při vytváření projektu je potřeba založit první pohled. V nabídce jsou možnosti založit prázdnou tabulku, prázdnou nástěnku, nebo použít jednu z připravených projektových šablon, které zároveň přednastaví úkolům atributy. V nabídce jsou šablony *Team backlog*, která vytvoří nástěnku vhodnou například pro Kanban, a *Feature*, který vytvoří tabulkový pohled a přidá do projektu iterace. [24]

Obrázek 3.1 Výběr šablony projektu [28]



Projektům lze nastavit jméno, popis a lze jim přidat README soubor definovaný ve formátu markdown. Pro projekty lze nastavit viditelnost. Veřejné projekty může vidět kdokoliv na internetu, soukromé projekty vidí jen uživatelé GitHubu, kteří mají přidělený přístup. [24]

3.1.1 Přístup k projektům

Pro projekty organizace může administrátor nastavit přístupová práva hromadně pro celou organizaci, nebo po týmech, nebo pro každého jedince zvlášť. Výchozí nastavení dává práva zápisu všem členům organizace. Toto nastavení lze změnit na jinou úroveň přístupových práv, nebo změnit tak, že členové organizace automaticky žádná přístupová práva k projektu mít nebudou.

Pro projekty uživatele lze přizvat do projektu jednotlivé uživatele, pro které se nastavují přístupová práva zvlášť.

Úrovně přístupových práv k projektu jsou:

Čtení Možnost zobrazit si projekt

Zápis Možnost zobrazit si a editovat projekt

Admin Možnost zobrazit projekt, editovat projekt a nastavovat přístupová práva uživatelům

[24]

3.2 Úkoly

V projektech lze sledovat a spravovat úkoly, návrhy úkolů a žádosti o sloučení změn kódu. Úkoly jsou *issues* nástroje GitHub Issues. Jsou vázány ke konkrétnímu repozitáři. Žádost o sloučení změn kódu se v prostředí GitHub nazývá pull request. Pull requesty jsou stejně jako úkoly vázány k repozitáři. Návrhy úkolů se vytváří v projektu a nejsou vázány ke konkrétnímu repozitáři. Návrh úkolu lze konvertovat na standardní úkol, při tomto úkonu je úkol přiřazen ke konkrétnímu repozitáři. Každou položku (úkol, návrh, pull request) lze archivovat, když už není aktuální.

V GitHub Project je omezení na celkový počet položek. Projekt může obsahovat maximálně 1200 aktivních položek a 10000 položek archivovaných. [24]

3.2.1 Atributy

Pro položky spravované v projektu lze nastavit libovolný počet atributů. Existující atributy je možné přejmenovávat a mazat. Atributy jsou těchto typů:

- Text
- Číslo
- Datum
- Výběr z výčtu hodnot
- Iterace
- Sleduje
- Sledováno

Typ *iterace* uchovává název iterace a data kdy iterace začíná a končí. Do typů *sleduje* a *sledováno* se zadávají reference na jiné úkoly. Lze tak vytvářet hiearchie úkolů. Tyto typy jsou v době psaní práce ve stavu soukromé betaverze. [24]

3.3 Pohledy

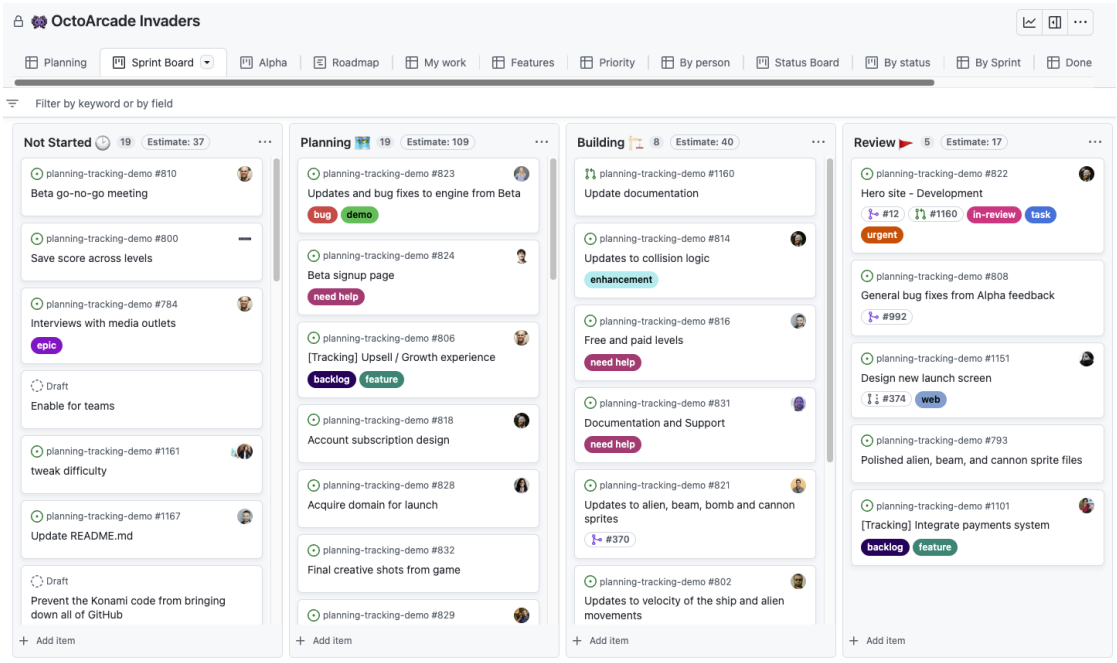
Pohled, *view*, poskytuje uživateli přehled nad projektem. Je to výchozí obrazovka pro řízení projektu v GitHub projects. Jedná se o přehled položek projektu, a to buď všech, nebo nebo jejich podmnožiny, pokud jsou v pohledu aplikovány filtry. [29]

Jakým způsobem jsou položky zobrazeny je dáno zvolený rozložením. GitHub Projets podporuje 3 druhy rozložení.

Tabulka. Pohled tohoto typu emuluje tabulkový procesor. Položky jsou zobrazeny pod sebou po jedné položce na řádek. Atributy položek jsou zobrazeny ve sloupcích. Které atributy budou v tabulce zobrazeny je nastavitelné. Položky v tabulce lze filtrovat, řadit a shlukovat do skupin podle zvolených kritérií. Kritéria se natavují podle hodnot atributů položek. Příklad pohledu tohoto typu je na obrázku 2.4. [29]

Nástěnka. Nástěnka zobrazuje položky ve sloupcích. Zařazení položky do sloupce je určeno podle hodnoty atributu zvoleného pro daný pohled. Může být použit atribut typu *výběr z výčtu hodnot*, nebo typu *iterace*. Při posunutí položky z jednoho sloupce do druhého se aktualizuje hodnota atributu podle sloupce, do kterého byla položka přesunuta. [29]

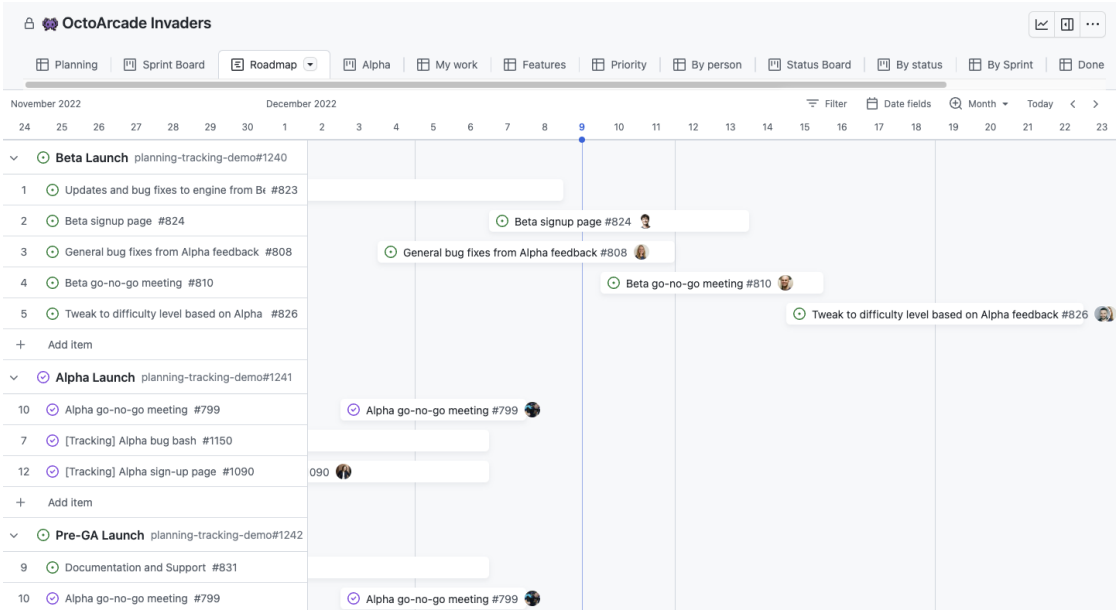
Obrázek 3.2 Pohled typu *nástěnka* [30]



Roadmap. Toto rozložení poskytuje pohled na projekt z vyšší úrovně. Zobrazuje časový úsek ve kterém jsou rozmístěny úkoly podle plánovaného data začátku práce a data na které je plánováno úkol dokončit. Úkoly je možné po pohledu posouvat a tím aktualizovat data. Časový interval a atributy použité jako data je možné libovolně nastavit.

Tento pohled je součástí uzavřené beta verze, není zatím veřejně dostupný. [29]

Obrázek 3.3 Pohled typu *roadmap* [31]



3.3.1 Filtry

Které položky budou zobrazeny se nastavuje pomocí filtrů. Filtrů může být zároveň aplikován libovolný počet, lze je kombinovat. Při aplikování více filtrů musí položka projít všemi filtry, podmínky všech filtrů jsou kombinovány pomocí logické funkce `AND`. Kombinace filtrů pomocí logické funkce `OR` není zatím podporováno[1].

Filtry jsou definovány textově. Vypadají jako dvojice, kde levý a pravý člen dvojice jsou odděleny znakem dvojtečky. Na levé straně je název atributu, nebo klíčové filtrovací slovo, na pravé straně je hodnota, nebo množina hodnot. Při psaní podmínek pro filtr funguje automatické doplňování, webová aplikace napovídá uživateli podmínky, jména atributů a možné hodnoty.

Kliknutí na hodnotu atributu zobrazenou u položky v pohledu typu *nástěnka* automaticky přidá do pohledu filtr pro zobrazení položek s touto hodnotou. Opětovné kliknutí na hodnotu filtr opět odstraní.

Například filtr `status:"In Progress"` ukáže všechny položky, jejichž stav, *status*, je nastaven na hodnotu *In Progress*. [29]

- Pro filtrování pomocí více hodnot je možné uvést hodnoty oddělené čárkou. Například filtr `label:"feature","bug"` vypíše všechny položky, označené štítkem jako *feature* nebo *bug*.
- Pro indikaci že atribut nenabývá nějaké konkrétní hodnoty lze použít znak `-` umístěný před filtr. Například `-label:bug` zobrazí pouze položky, které nejsou označeny jako *bug*.
- Pro indikaci absence jakékoliv hodnoty u atributu je možné zadat `no:` a potom jméno atributu. Například filtr `no:assignee` zobrazí položky, které nemají přiděleného žádného uživatele (*assignee*)
- Pro filtrování podle stavu položky je možné pomocí `is:` a názvu stavu.
- Pro použití více filtrů zároveň stačí uvést filtry oddělené mezerou. Například pro zobrazení všechny položek, které jsou rozpracované, nejsou označené jako *bug* a nemají přiděleného uživatele lze použít filtr `status:"In progress" -label:"bug" no:assignee`

[1] Platí v době psaní této práce, tzn. prosinec 2022

- Pro filtrování podle současné iterace, nebo iterace předcházející, či následující po té současné lze použít hodnoty *@previous*, *@current*, *@next*. Například `iteration:@current`. Tento filtr zobrazí položky z právě probíhající iterace.
- Pro zobrazení položek přidělených uživateli se použije hodnota *@me*. Jako příklad lze uvést `assignee:@me`. Tento filtr zobrazí všechny položky, přiřazené uživateli, který si v dané chvíli zobrazuje pohled.
- Pro filtrování podle data poslední aktualizace položky je možné použít `last-updated:` a počet dní. Filtr podporuje jako hodnoty pouze {*number*}*days*, nebo *1day*. Jako příklad je možné uvést `last-updated:7days`. Takový filtr zobrazí položky, které byly aktualizovány 7 dní zpět, nebo ještě dříve.
- Pro atributy s číselnou hodnotou nebo datem je možné používat operátory `>`, `=>`, `<`, `<=` pro hodnotu atributu větší, větší nebo rovnou, menší, menší nebo rovnou uvedené hodnotě. Například `points:>5` zobrazí jen ty položky, jejichž hodnota atributu *points*, body, je větší než 5.
- Operátor `..` lze použít pro rozsah. `target:2022-03-01..2022-03-15` ukáže jen položky s hodnotou atributu *target*, cíl, mezi 1. březnem a 15. březnem. Jedná se o uzavřený interval.

Stejné filtry jako pro pohledy lze použít i pro grafy popisované v sekci 3.5. [29]

3.4 Automatizace

GitHub Projects poskytuje několik způsobů jak automatizovat procesy řízení průběhu projektu. Většinou se jedná o aktualizace dat v položkách v závislosti na události projektu. [32]

3.4.1 Vestavěná automatizace

V nástroji Projects jsou vestavěná workflow, pomocí kterých lze automaticky aktualizovat stav položek. Tato workflow se zapínají a nastavují ve webovém uživatelské rozhraní GitHub Projects.

U všech workflow je možné nastavit jaký status se má u položky nastavit. U některých workflow je možné nastavit typ položky, zda se má automatická akce provést pro úkol, nebo pull request. U workflow týkajících se událostí na pull requestu tato možnost logicky není.

Automatickou aktualizaci stavu položky lze nastavit pro následující události:

- Položka přidána do projektu
- Položka znovu otevřena
- Položka zavřena
- Vyžádány změny pro pull request
- Pull request schválen
- Pull request sloučen

Ve stavu veřejné beta verze[2] je workflow pro automatickou archivaci položek. Zde lze pomocí filtru (viz 3.3.1) specifikovat jaké položky mají být archivovány.

Po vytvoření nového projektu jsou zapnuta 2 workflow automaticky. První workflow je o tom, že když je úkol nebo pull request zavřen, jeho status je nastaven na *Done*, hotovo. A druhé workflow je, že když jsou změny pull requestu sloučeny do cílové větve kódu, tak se status pull requestu v Projects také nastaví na *Done*. [32]

[2] V prosinci 2022

Obrázek 3.4 Obrazovka nastavení vestavěné automatizace

3.4.2 GraphQL API

GraphQL je dotazovací jazyk pro API a serverové běhové prostředí pro provádění dotazů definovaných v tomto jazyce. GraphQL služba je tvořena definicí typů a jejich atributů a poskytuje funkce pro získání jakýchkoliv z těchto atributů u definovaných typů. Příklad definice je ve výpisu kódu 1. GraphQL umožňuje v dotazech definovat přesně která data a atributy mají být vráceny. [33]

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

Výpis kódu 1 Příklad definice typů a atributů v GraphQL [33]

GitHub nabízí možnost pracovat s daty na jeho platformě pomocí GraphQL API. Pomocí tohoto rozhraní je možné spravovat i projekty. Lze vytvářet projekty, získávat o nich informace a manipulovat s položkami projektu. Toho je možné využít například ve workflow GitHub Actions pro automatickou aktualizaci projektu, což je rozebráno v sekci 3.4.3.

Z příkazové řádky lze toto API zavolat pomocí GitHub CLI aplikace. Také je možné pro volání API využít HTTPS protokol. Na adrese `https://api.github.com/graphql` je umístěn endpoint, který příjmá POST požadavky s GraphQL dotazy. [32]

GitHub také podporuje nastavení webhooku, mechanizmu, který při vybrané události pošle na uživatelem nastavenou adresu HTTP POST požadavek. V těle požadavku jsou data o události ve formátu JSON.

Pro Projects je možné nastavit webhook, který pošle požadavek, když se provede akce s položkou v projektu (vytvoření, smazání, úprava, archivace atd.). Webhook je možné zavést i pro události týkající se pull requestů a úkolů v rámci Issues, které nejsou přímou součástí Projects, ale úzce s položkami projektu souvisí. [34]

Využití webhooků a GraphQL API pro správu projektů dává uživatelům GitHubu možnost naprogramovat si vlastní aplikaci pro automatizaci. Webhook se nastaví, aby posílal požadavek na server, kde běží aplikace, která zpracuje informace o události a na jejich základě provede pomocí GraphQL API GitHubu změny. [32]

Získání ID projektu

Pro úpravu projektu pomocí GraphQL API je potřeba znát ID projektu. To může uživatel také získat pomocí API. Je k tomu potřeba znát jméno organizace nebo uživatele, kterému projekt patří a číslo projektu. Číslo projektu uživatel zjistí z URL projektu. Například pro projekt s adresou `https://github.com/orgs/octo-org/projects/5` je to číslo 5.

Ve výpisu kódu 2 je zobrazen dotaz na získání ID projektu pomocí GitHub CLI. Za řetězec *ORGANIZATION* se zamění jméno organizace a za *NUMBER* číslo projektu.

```
gh api graphql -f query='
  query{
    organization(login: "ORGANIZATION"){
      projectV2(number: NUMBER) {
        id
      }
    }
  }'
```

Výpis kódu 2 GraphQL dotaz na ID projektu pomocí GitHub CLI [32]

Ve výpisu kódu 3 je ten samý dotaz jako HTTP požadavek poslaný pomocí programu cURL. Za `TOKEN` je potřeba zaměnit přístupový token GitHubu, který má povolený přístup k projektu. Z příkazu je vidět, že se jedná POST požadavek na adresu GitHub API. Samotný GraphQL dotaz, specifikující data, která chceme získat, je v těle požadavku. [32]

```
curl --request POST \
  --url https://api.github.com/graphql \
  --header 'Authorization: Bearer TOKEN' \
  --data '{"query":"query{organization(login: \"ORGANIZATION\")\
          {projectV2(number: NUMBER){id}}}"}'
```

Výpis kódu 3 GraphQL dotaz na ID projektu pomocí cURL [32]

Příklad odpovědi na tyto požadavky je ve výpisu kódu 4. Jsou to vyžádaná data ve formátu JSON.

```
{
  "data": {
    "organization": {
      "projectV2": {
        "id": "PVT_kwDOBi2Qk84ABQPD"
      }
    }
  }
}
```

Výpis kódu 4 Příklad odpovědi na GraphQL dotaz na ID projektu

Nastavení hodnoty atributu položky

Pomocí GraphQL API je možné v projektu dělat změny, například nastavit hodnotu atributu položky projektu. Je k tomu potřeba znát ID projektu, ID položky a ID atributu.

Příklad dotazu je ve výpisu kódu 5. Za všechna ID se je potřeba dosadit konkrétní hodnoty a za *"Updated text"* je možné dosadit libovolnou hodnotu, kterou chce uživatel nastavit.

```
mutation {
  updateProjectV2ItemFieldValue(
    input: {
      projectId: "PROJECT_ID"
      itemId: "ITEM_ID"
      fieldId: "FIELD_ID"
      value: {
        text: "Updated text"
      }
    }
  ) {
    projectV2Item {
      id
    }
  }
}
```

Výpis kódu 5 GraphQL dotaz na aktualizaci hodnoty atributu položky [32]

3.4.3 Automatizace pomocí GitHub Actions

GitHub Actions je produkt GitHubu, který umožňuje definovat procesy, workflow, které se automaticky provedou po zvolené události v repozitáři. Mezi události patří například přidání nového kódu, vytvoření pull requestu, uzavření pull requestu, změny v nastavení repozitáře a další.

Workflow je možné definovat v konfiguračních souborech formátu YAML umístěných ve složce `.github/workflows` v kořenovém adresáři repozitáře. Proces poběží na dedikovaném virtuálním stroji, inicializovaném pro každý běh, který je zvolený v YAML souboru. K dispozici jsou dispozici stroje s operačnímy systémy Windows, Ubuntu Linux a macOS i Docker kontejnery. Navíc je možné si nakonfigurovat vlastní prostředí pro spouštění workflow na vlastním serveru s libovolným operačním systémem. [35]

Ve workflow lze spouštět příkazy příkazové řádky na stroji provádějícím toto workflow. K tomu je možné ještě spouštět *akce*, což jsou komplexní příkazy, které lze v různých workflow volat opakovaně. Je možné definovat své vlastní akce nebo použít akce z GitHub Marketplace[3].

Ve výpisu kódu 5 je příklad definice workflow. Workflow je spuštěno po tom, co jsou změny kódu nahrány do repozitáře na GitHubu (`git push`) a běží na Ubuntu. Actions stáhne nový kód, nainstaluje testovací framework Bats a spustí příkaz pro vypsání verze frameworku. [35]

Mezi událostí po kterých lze spustit workflow nepatří žádné události týkající se Projects, to bylo možné pouze u starší verze Projects (Classic), ale je možné v rámci workflow provést změny na projektu pomocí GitHub GraphQL API. Například po vytvoření pull requestu lze touto cestou vytvořit nový úkol v projektu pro zkontrolování pull requestu. [32]

[3] GitHub Marketplace je místo kde mohou uživatelé sdílet akce

```
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

Výpis kódu 6 Definice workflow GitHub Actions [35]

3.5 Grafy

Projects umožňuje tvoření vhledů do projektu, *insights*, což jsou diagramy, grafy, zobrazující data z projektu. Jsou 2 druhy grafů: „současné“ grafy a historické grafy. Historické grafy jsou dostupné pouze pro organizace platící plán GitHub Team a organizace používající GitHub Enterprise Cloud.

Historické grafy zobrazují data v časovém úseku, na ose X (svislá osa) mají čas. Současné grafy mají na ose X zvolený atribut položek projektu. Na ose Y mají oba typy grafů zvolený atribut položek projektu. Na ose X lze volitelně položky seskupit podle vybraného atributu.

Projects podporuje tyto typy grafů:

- Pruhový
- Sloupcový
- Spojnicový
- Plošný
- Skládaný pruhový
- Skládaný sloupcový

Pro upřesnění položek, které jsou použity pro vykreslení grafu je možné používat filtry stejně jako u pohledů. Filtry jsou popsány v sekci 3.3.1. [36]

Obrázek 3.5 Ilustrační graf [37]

3.6 Doporučené postupy

GitHub v dokumentaci k projektům doporučuje následující postupy a praktiky pro efektivní řízení projektů.

Rozdělování větších položek na menší

Rozdělování větších položek na menší umožňuje rozdělit práci na jednodušší úkoly, umožňuje paralelní práci více členů týmu a vede k menším pull requestů, které je jednodušší zkontrolovat.

Pro sledování menších položek tvořících jeden větší celek je možné použít štítky, což je jeden z možných atributů položek, nebo *tasklist*, seznam úkolů. Seznam úkolů lze vytvořit v GitHub Issues, když je v popisu úkolu ve formátu Markdown uveden seznam, kde je před položkami seznamu napsáno `[ ]` pro nesplněný úkol, nebo `[x]` pro splněný úkol. Ve výpisu kódu 7 je příklad zdrojového kódu takového seznamu. [38]

```
- [x] #739
- [ ] https://github.com/octo-org/octo-repo/issues/740
- [ ] Add delight to the experience when all tasks are complete :tada:
```

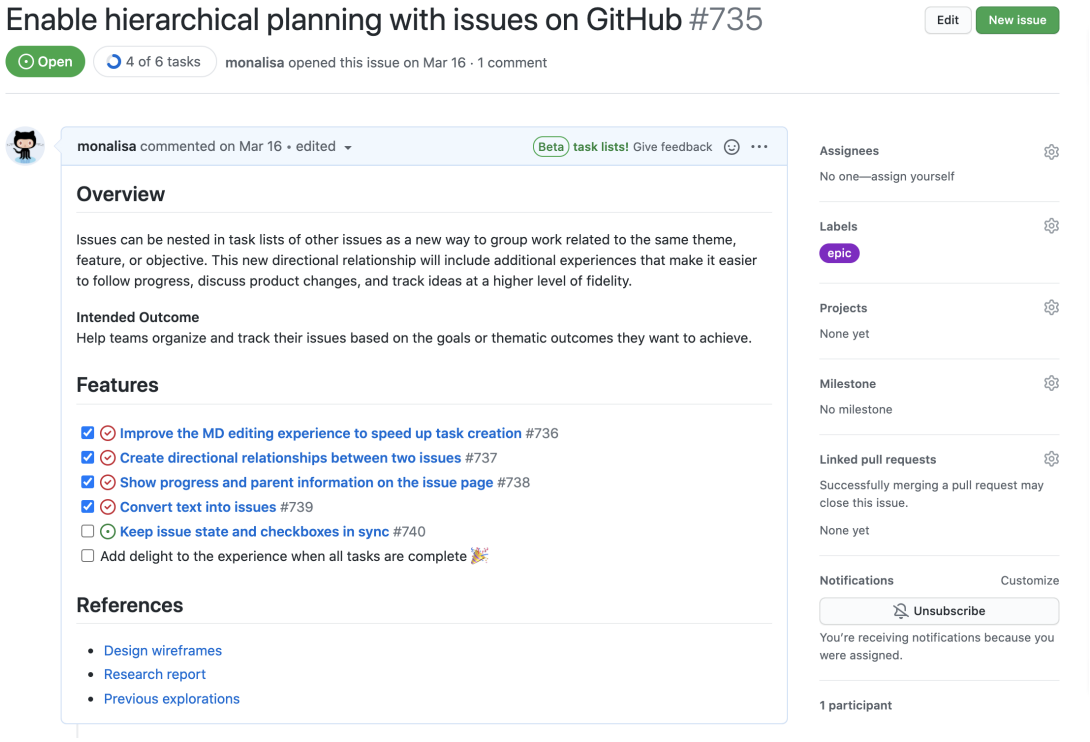
Výpis kódu 7 Zdrojový kód seznamu [35]

Při zobrazení issue s seznamem úkolů potom webové UI GitHubu přidá ikony pro splněné a nesplněné úkoly, hypertextové odkazy pro úkoly specifikované URL, nebo číslem úkolu a počet splněných úkolů. Příklad tohoto zobrazení je na obrázku 3.6. [38]

Komunikace

Položky obsahují vestavěné funkce pro usnadnění komunikace mezi členy týmu. Lze používat zmínky ve tvaru `@username` pro upozornění členů týmu na komentář. Lze přidělovat členy týmu

Obrázek 3.6 Vykreslení issue se seznamem úkolů [39]



k položkám pro komunikaci zodpovědnosti za položky. Je možné provázat odkazy související položky aby bylo jasnější, které spolu souvisí. [38]

Použití popisu projektu a README

Vyplnění popisu projektu a přidání souboru README usnadní přístup k informacím o projektu. Je vhodné například uvést cíl projektu, popsat pohledy a jejich použití, uvést relevantní odkazy a konktakty.

Soubor README podporuje formát Markdown, takže je možné vložit obrázky a použít pokročilé formátování jako jsou nadpisy, odkazy a seznamy. [38]

Použití pohledů

Použití pohledů dá uživatelům vhled do projektu z různých úhlů. Je možné:

- Filtrovat podle stavu pro zobrazení všech položek, na kterých ještě nikdo nezačal pracovat
- Seskupit položky podle priority pro monitorování objemu práce s vysokou prioritou
- Seřadit položky podle data ukáže, kterým položkám se nejrychleji blíží datum vydání

[38]

Jeden zdroj pravdy

Pro udržení integrity dat je vhodné udržovat jeden zdroj pravdy. Například plánované datum vydání je vhodné udržovat na jednom místě, místo toho, aby bylo uvedeno ve více atributech

více položek. Pokud ho pak bude potřeba změnit, bude stačit to udělat na jednom místě. Pokud je potřeba informaci referencovat z více míst, je možné použít odkaz.

Projects automaticky aktualizuje položky podle dat z repozitáře a z Issues. Pokud je upraven přidělený uživatel nebo štítek úkolu, změna se automaticky projeví a ukáže v položce projektu. [38]

Použití automatizace

Použití automatizace umožní strávení méně času na rutinních úpravách a strávení více času prací na projektu. Čím méně věcí je potřeba si pamatovat dělat manuálně, tím větší šance je, že projekt bude vždy aktuální. [38]

Použití různých typů atributů

Je vhodné používat různé typy atributů položek podle potřeb projektu. Typ *iterace* umožní seskupit položky podle iterací a porovnat pracovní nálož jednotlivých iterací. Dále umožní nastavit pohled na průběh práce v aktuální iteraci, dá týmu přehled nad dokončenými položkami v jiných iteracíseskupit položky podle iterací a porovnat pracovní nálož jednotlivých iterací. Dále umožní nastavit pohled na průběh práce v aktuální iteraci, dá týmu přehled nad dokončenými položkami v jiných iterací.

Použití atributu typu *výběr z položek* umožní specifikovat informace jako je priorita nebo fáze projektu. Pomocí tohoto atributu opět jednoduše seskupit položky v pohledech, protože může nabývat jen předdefinovaných hodnot. [38]

Kapitola 4

Řízení projektu v GitHub Projects

4.1 Analýza řízení projektů ve firmě

Před samotným použitím GitHub Projects je nejdříve popsáno řízení projektů v softwarové firmě. Jsou zde popsány postupy, nástroje a metodika, které se ve zkoumané firmě používají. Tyto postupy jsou implementovány v GitHub Projects, aby byly otestováno vlastnosti a možnosti tohoto nástroje.

Ve firmě se metodika a nástroje liší podle konkrétního projektu. Pro potřeby práce byl vybrán jeden z těchto projektů. Metodika a nástroje v něm použité se ale používají v mnoha dalších projektech firmy. Z technického hlediska je projekt je webová aplikace, jejíž kód je rozdělen na 2 části. Backendovou část, což je část aplikace, běžící na serveru, a frontendovou část, což je část aplikace, běžící u klienta ve webovém prohlížeči.

4.1.1 Metodika

Pro řízení projektu je použita metodika Scrum. Vývoj probíhá ve Sprintech trvajících 3 týdny, ve kterých se pracuje na úkolech, které jsou vybírány z backlogu. Vývojový tým tým má 7 členů, týmu pomáhá scrum master a úkoly do backlogu podle požadavků zákazníka přidává product owner. Každý člen týmu má roli buď frontendového vývojáře, backendového vývojáře, nebo testera. I když se členové dokáží v případě potřeby do jisté míry navzájem zastoupit, každý člen dělá primárně úkoly spadající pod jeho roli.

4.1.2 Nástroje

Pro řízení vývoje se používá nástroj Jira. Zdrojový kód je uložen ve dvou Git repozitářích, jeden pro backend, jeden pro frontend, které jsou umístěny na firemním GitLab serveru. GitLab se také používá na code review, které se provádí po vytvoření merge requestu (žádosti o přidání změn kódu do původní větve).

4.1.3 Události Scrumu

Sprint. V průběhu si vývojáři přiřazují jednotlivé úkoly. Když na nich pracují, tak aktualizují jejich stav. Používá se Sprintová nástěnka v nástroji Jira, *Sprint Board*, a detailní pohled na

úkol. Během Sprint tým monitoruje průběh na Sprintové nástěnce a ještě pomocí grafu zvaného Burndown.

Plánování Sprintu. Obsah Sprintu se před jeho začátkem vždy dohodne na plánovací schůzce. Úkoly do Sprintu se vyberou z Product Backlogu, kam Product Owner úkoly přidává a nastavuje jim prioritu. Pro tuto činnost se používá pohled Backlog v nástroji Jira. V něm jsou zobrazeny kromě úkolů backlogu zobrazeny i plánované Sprint Backlogy.

Backlog Refinement. Během pravidelného meetingu pojmenovaného Backlog Refinement tým prochází úkoly v Product Backlogu, analyzuje je, upravuje a rozděluje. Při této činnosti se používá pohled Backlog v nástroji Jira a pohled na detail úkolu.

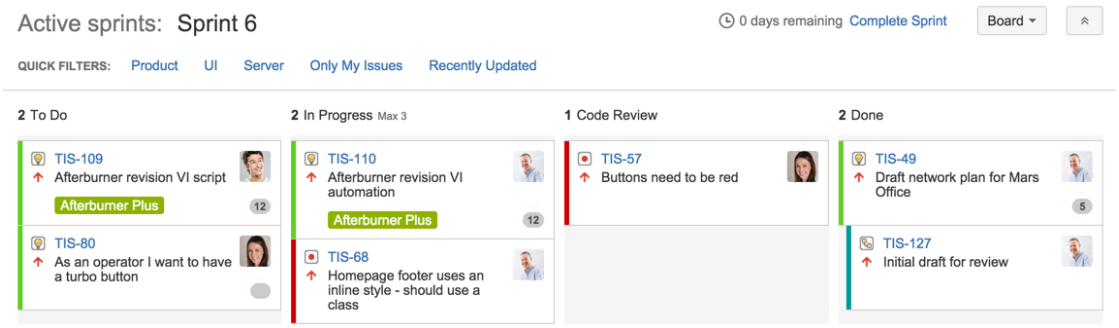
Sprint Review. Na konci Sprintu tým prezentuje výsledky manažerovi, který je hlavním stakeholderem a sbírá feedback. Feedback je zapisován buď nejdříve do zvláštního souboru, nebo když je to vhodné přímo k úkolům v nástroji Jira.

Sprint Retrospective. Sprint je uzavřen meetingem Sprint Retrospective, kde tým probírá jak se povedlo Sprint dokončit. Zde se hojně používá Burndown graf, Sprintová nástěnka a pro přesun úkolů, které se nepovedly dokončit zpět do Product Backlogu se používá pohled Backlog.

4.1.4 Sprintová nástěnka

Sprintová nástěnka je zobrazena jako tabulka kartiček s úkoly, které jsou seřazeny do sloupců podle svého stavu. Kartičky lze mezi sloupci přesunovat a aktualizovat tak jejich stav. Na kartičkách je vidět jméno úkolu, druh úkolu, odhad ve Story Points a přiděleného uživatele.

Obrázek 4.1 Ilustrační Sprintová nástěnka v nástroji Jira [40]



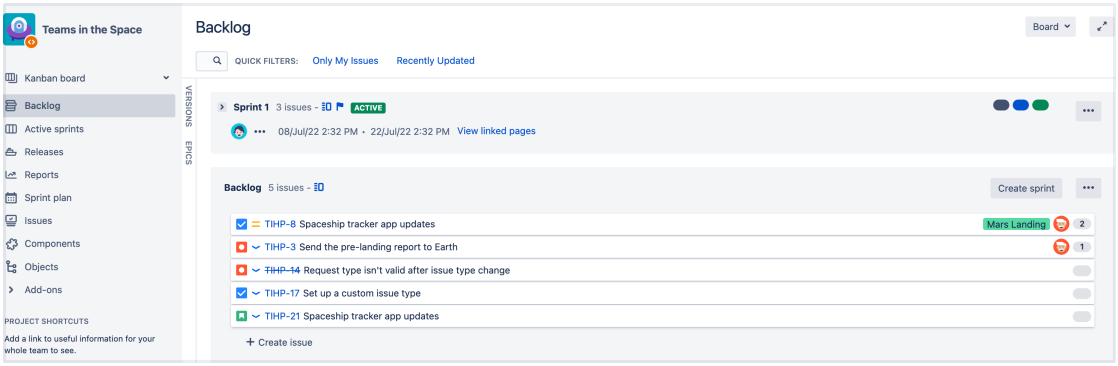
4.1.5 Produktový backlog ve firmě

Product Backlog je uchováván v nástroji Jira pomocí pohledu funkcionality *Backlog*. U každého úkolu je jeho jméno, popis, akceptační kritéria, druh (zda se jedná o user story nebo bug), číslo Sprintu, odhad ve story pointech (TODO zde přidat referenci na sekci s vysvětlením), stav, kdo úkol vytvořil a přidělený člen týmu, který na úkolu má pracovat. Jira každý úkol označí unikátním deskriptorem. Ten se skládá z kódu projektu a čísla úkolu v projektu.

Při vytváření úkolu jsou povinné údaje jméno, popis a druh. Stav úkolu, vytvářející uživatel a přidělený uživatel jsou nastaveny automaticky na inicializační hodnoty. Stav na hodnotu otevřený, údaje o uživatelích jsou oba nastaveny na uživatele, který úkol vytvořil.

Backlog je v nástroji Jira znázorněn vertikální seznam úkolů. V tomto seznamu je u úkolů vidět pouze název, přiřazený identifikační kód a odhad ve Story Points. Pro zobrazení všech informací o úkolu je potřeba otevřít pohled s detailem úkolu. V pohledu *Backlog* jsou nad Product Backlogem zobrazeny Sprint Backlogy plánovaných Sprintů. V tomto seznamu můžeme úkoly myší přesouvat a měnit pořadí. Úkoly zde můžeme přesouvat i do Sprintových backlogů a tím úkoly řadit do Sprintů. (ilustrační obrázek)

Obrázek 4.2 Ilustrační Backlog pohled v nástroji Jira [41]



4.1.6 Workflow úkolů

Možné stavy úkolů jsou: *otevřený*, *probíhá*, *review*, *testuje se*, *hotový*. Typicky si úkol projde stavy v tomto pořadí. Ze stavů *review* a *testuje se* se úkol často může vrátit do stavu *probíhá*, pokud je nalezen nějaký problém.

4.1.7 Automatizace

Nejvíce se automatizace využívá u integrace s GitLabem a točí se okolo merge requestů. Pokud je vytvořen merge request a úkol je ve stavu *probíhá*, je stav automaticky aktualizován na hodnotu *review*. Ve chvíli kdy je merge request uzavřen a kód se změnami je sloučen do původní větve, se stav úkolu přepne z *review* na *testuje se*. Po sloučení je totiž aktualizovaná větev nasazena na testovací prostředí. Aby tato integrace mezi nástroji Jira a GitLab fungovala, je potřeba, aby jméno Git větve s změnami začínalo deskriptorem úkolu z nástroje Jira.

Další prvek automatizace je implementován při posunutí úkolu ze stavu *otevřený* do stavu *probíhá*. Uživatel, který tuto změnu udělá je k úkolu automaticky přiřazen.

4.2 Implementace v GitHub Projects

Celý projekt využívá infrastrukturu a nástroje GitHub běžící v cloudu společnosti GitHub. Pro řízení projektu slouží sada nástrojů Issues, hlavně nástroj Projects. Pro hosting Git repozitářů slouží samotný GitHub, neboť je je to jeho původní a hlavní služba, od které se odvíjí všechny ostatní služby. Pro CI/CD, tedy automatické sestavení, testování a nasazení se používá nástroj Actions.

Projekt se jmenuje *Thesis Project*. Má tři repozitáře kódu, *thesis-backend*, pro kód běžící na serveru a *thesis-frontend* pro kód běžící v browseru. Nástroj Issues vyžaduje, aby úkoly byly přiřazeny konkrétnímu repozitáři, bez repozitáře mohou být pouze návrhy v nástroji Projects. Některé úkoly jsou ale komplexnější a vyžadují změny v obou repozitářích. Pro správu takových úkolů je vytvořen tetí repozitář *thesis-epics*. Pokud by nebyl použit zvláštní repozitář, tak by tyto úkoly museli buď zůstávat nastaveny jako návrhy, nebo by byly přiděleny jen jednomu z repozitářů.

Projekt i repozitáře jsou vytvořeny v organizaci *Warzeada-Thesis-Org*. Pro projekty vytvořené samostatnými uživatelem a nikoliv organizací nelze nastavit webhook spouštěný po aktualizaci položky projektu. Organizace, to znamená i projekt a repozitáře, je dostupná na adrese `https://github.com/Warzeada-Thesis-Org`.

4.2.1 Úkoly

Úkoly jsou implementovány pomocí úkolů z nástroje Issues. Úkoly v Issues automaticky obsahují automaticky pole pro název, popis, přidělené uživatele, štítky (*labels*), pull request a milníky. U každého úkolu je uvedeno, kdo ho vytvořil.

V GitHubu jsou k dispozici výchozí štíky jako *bug*, *enhancement*, *duplicate*, *question*, *invalid*, atd. Je možné přidat přidat libovlné vlastní štítky. Štítků může být nastaveno na jedné položce i několik. [42]

V nástroji Projects jsou pro položky doplněny tyto atributy:

Status typ *výčet*, stav úkolu, automaticky přidáno v každém projektu nástrojem Projects

Sprint typ *iterace*, údaj o tom, do kterého Sprintu je úkol naplánován

Story Points typ *číslo*, odhad pracnosti úkolu

Acceptace Criteria typ *text*, co musí být splněno, aby mohl být úkol považován za splněný

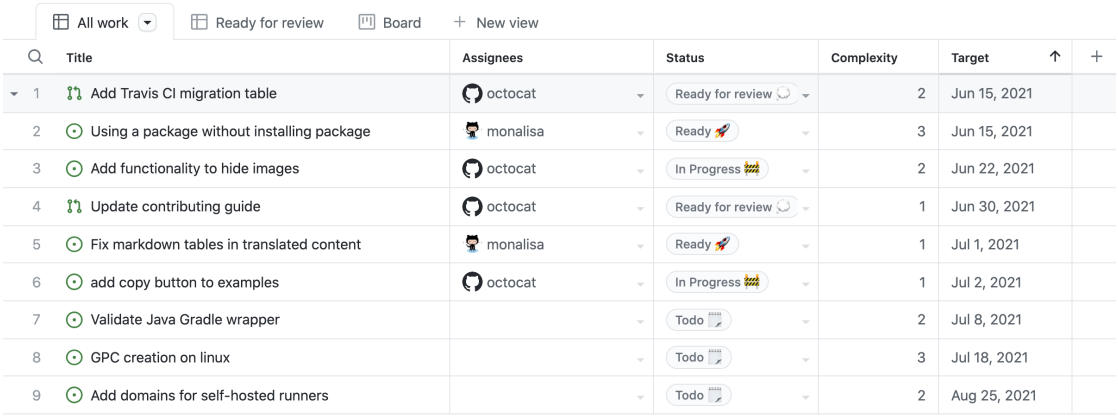
Pull request lze přiřadit k úkolu buď manuálním nastavením přes v UI, nebo automaticky uvedením jednoho z klíčových slov, *fix*, *close*, *resolve*, následovaného číslem úkolu, nebo odkazem na úkol v popisu pull requestu. [42]

4.2.2 Produktový backlog

Produktový backlog je implementován jako tabulkový pohled na všechny úkoly. V pohledu je nastaveno, aby se pro úkoly zobrazoval Sprint, stav, přidělení uživatelé, informace o odhadech (atribut *Story Points*) a štítky. Úkoly jsou seskupeny podle Sprintu. Díky tomu vzniká přehled o plánovaných Sprintech a úkolech v nich. Úkoly bez Sprintu představují Produktový backlog. Skupiny úkolů představují Spring backlogy. V tomto pohledu lze vytvářet nové úkoly a přidávat je do Sprintů. Metodou *drag and drop*[1] je možné přesouvat úkoly mezi Sprinty. Zobrazené hodnoty atributů úkolů je možné upravovat.

[1] Úkol se musí myší "uchopit"za číslo

Obrázek 4.3 Produktový backlog v GitHub Projects [26]



4.2.3 Sprintová nástěnka

Sprintová nástěnka je pohled typu nástěnka, kde jsou úkoly seskupeny podle stavu. Je aplikován filtr `sprint:@current`, na nástěnce jsou tak jen úkoly, které jsou zařazeny do současného Sprintu. V pohledu je nastaveno, aby na kartičkách představujících úkoly bylo zobrazeno číslo úkolu, jméno úkolu, přiřazený uživatel, štíky pro indikaci druhu úkolu a odhad. Přesouváním úkolů mezi sloupci lze aktualizovat jejich stav.

4.2.4 Automatické sestavení a nasazení

Automatizace otestování, sestavení a nasazení je implementováno ve workflow GitHub Actions. Konfigurace je v souboru `ci.yml` ve složce `.github/workflows/`.

Při nahrání nového commitu do repozitáře na platformě GitHub automaticky spustí testy a sestavení. Při přidání nových změn do větve *main*, typicky po sloučení změn z větve s novým kódem, se spustí sestavení, testy a nasazení. Nasazení je pouze simulováno. Kód pro konfiguraci nasazení je ve výpisu kódu 8. Po doběhnutí nasazení se pomocí GitHub GraphQL API aktualizuje stav úkolu. Toto je popsáno v sekci 4.2.5.1.

```
# Simulated deployment job
deploy:
  if: github.ref == 'refs/heads/main'
  runs-on: ubuntu-latest
  needs: build
  steps:
    - name: Download the built jar file
      uses: actions/download-artifact@v3
      with:
        name: ${{ env.JAR_NAME }}
    - name: Deploy
      run: ls -lh
```

Výpis kódu 8

4.2.5 Automatizace

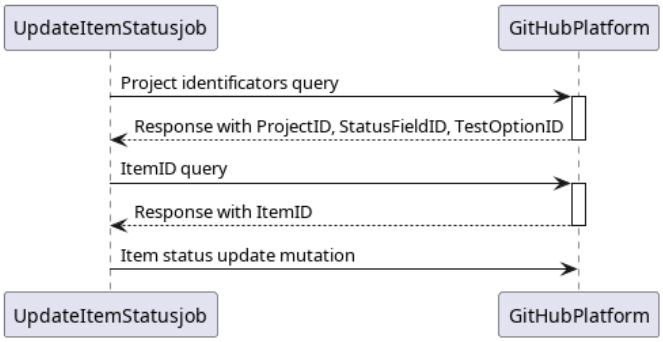
Pro projekt jsou zapnutá vestavěná workflow pro automatické nastavení stavu projektové položky na výchozí hodnotu *Todo* při vytvoření, nebo znovuotevření úkolu a nastavení na hodnotu *Closed* při uzavření úkolu. Pro komplexnější automatické procesy je využito GitHub GraphQL API, použití je popsáno v následujících dvou sekcích.

4.2.5.1 Změna stavu po nasazení

Aktualizace stavu úkolu po úspěšném nasazení je realizována zavoláním GitHub GraphQL API přímo z GitHub Actions workflow po úspěšném dokončení jobu pro nasazení. Pro volání API je využita aplikace příkazové řádky GitHub. Aby tato automatizace fungovala je potřeba, aby se větev se změnami mělo jméno začínající znakem # a číslem úkolu, například *#25.automatic.build*. Díky tomu je možné určit, ke kterému úkolu se změna váže.

Tato část je implementována v jobu *update-item-status*. Jsou potřeba celkem tři volání API. První pro získání identifikátorů, ID, projektu, datového pole stavu a stavové hodnoty *Test*. Druhé pro získání ID úkolu podle čísla úkolu parsovaného z názvu větve. A třetí pro samotnou aktualizaci stavu, která lze provést až po získání všech zmíněných ID. Tento proces je možné vidět v sekvenčním diagramu na obrázku 4.4.

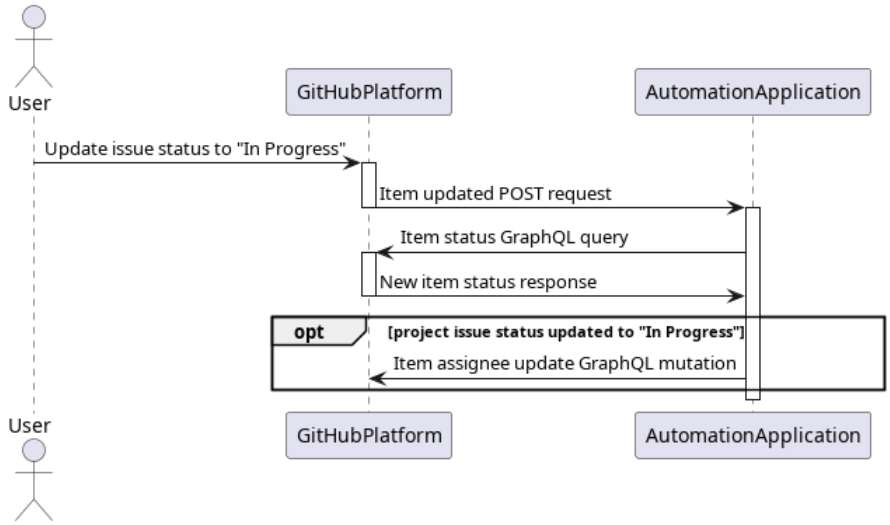
Obrázek 4.4 Sekvenční diagram aktualizace stavu položky



4.2.5.2 Aktualizace uživatele při přesunutí úkolu do stavu *Probíhá*

Automatická změna přiřazeného uživatele je realizována pomocí webhooku a volání GitHub API z vlastní aplikace. Při aktualizaci položky projektu je poslán z platformy GitHub POST požadavek na nastavenou adresu, na které příjmá požadavky aplikace. Aplikace ověří, jestli se jedná o úkol z projektu, který byl posunut do stavu *In Progress*. Pokud jedná, tak pošle přes GitHub GraphQL API požadavek na aktualizaci položky. Uživatel, který změnu provedl je tak přidělen k úkolu. Informace o novém stavu není obsažena v POST požadavku poslaném z webhooku, takže aplikace musí pomocí GraphQL API tuto informaci získat. Tento postup je naznačen v sekvenčním diagramu na obrázku 4.5.

Obrázek 4.5 Sekvenční diagram aktualizace uživatele po posunutí úkolu do stavu *Probíhá*



Aplikace je implementována v jazyce Java 17. Použité frameworky a technologie:

Spring Framework 6.0 aplikační framework poskytující Dependency Injection

Spring Boot pro automatickou konfiguraci pro Spring Framework

Spring Web pro zpracování příchozích HTTP požadavků

Spring WebFlux pro GraphQL klienta

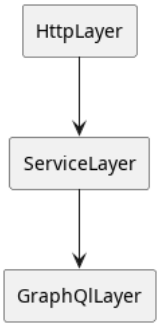
JUnit framework pro jednotkové testování

Mockito framework pro „mocking“

Gradle nástroj pro automatické sestavení projektu

Architektura aplikace je inspirována třívrstvou architekturou podnikových aplikací s tím rozdílem, že místo databázové vrstvy má aplikace GraphQL vrstvu. *HTTP* vrstva, umístěná v Java balíčku *rest*, zpracovává příchozí požadavky. Data z těchto požadavků posílá do *Service* vrstvy v balíčku *service*, kde se nachází logika aplikace. Vrstva *Service* potom přes pomocí vrstvy *GraphQL* v balíčku *graphql* pracuje s daty o projektu, uloženými na platformě GitHub.

Obrázek 4.6 Jednoduchý diagram architektury aplikace pro automatizaci



Spring WebFlux umožňuje uložit každý GraphQL zvlášť do svého souboru ve složce `src/main/resources/graphql-documents/` a ve zdrojovém kódu pak dotazy použít jen pomocí jména souboru.

```
mutation addAssignee(
    $issueId: ID!,
    $assigneeId: ID!,
    $clientMutationId: String
) {
    addAssigneesToAssignable(
            input: {
                assignableId: $issueId
                assigneeIds: [$assigneeId]
                clientMutationId: $clientMutationId
            }
    ) {
        clientMutationId
    }
}
```

Výpis kódu 9 GraphQL mutace pro přidání uživatele k úkolu

```
graphQlClient.documentName("addAssignee")
    .variable("issueId", issueId)
    .variable("assigneeId", assigneeId)
    .variable("clientMutationId", assigneeId)
    .retrieve("addAssigneesToAssignable.clientMutationId")
    .toEntity(String.class)
    .block(Duration.ofSeconds(maxBlockTime));
```

Výpis kódu 10 Volání GraphQL uloženého v souboru *addAssignee.graphql*

Aplikaci je třeba spustit na stroji, na který lze poslat požadavek z internetu. Aplikace přijímá požadavky typu POST na portu 8080, na cestě `/update`. Aplikace používá externalizovanou konfiguraci. Výchozí konfigurace je v repozitáři souboru *application.properties*. Při spouštění aplikace sestavené do *jar* souboru je automaticky prohledána složka, ve které se soubor nachází, a pokud se v ní nachází soubor *application.properties*, přepíše se jeho obsahem výchozí konfigurace.

Zdrojový kód aplikace se nachází na přiloženém mediu a ve veřejném repozitáři *thesis-backend*. Jeho součástí jsou jednotkové testy pro *Service* vrstvu a definice GitHub Actions workflow.

4.3 Simulace projektu

4.3.1 Plánování

Dlouhodobé plánování

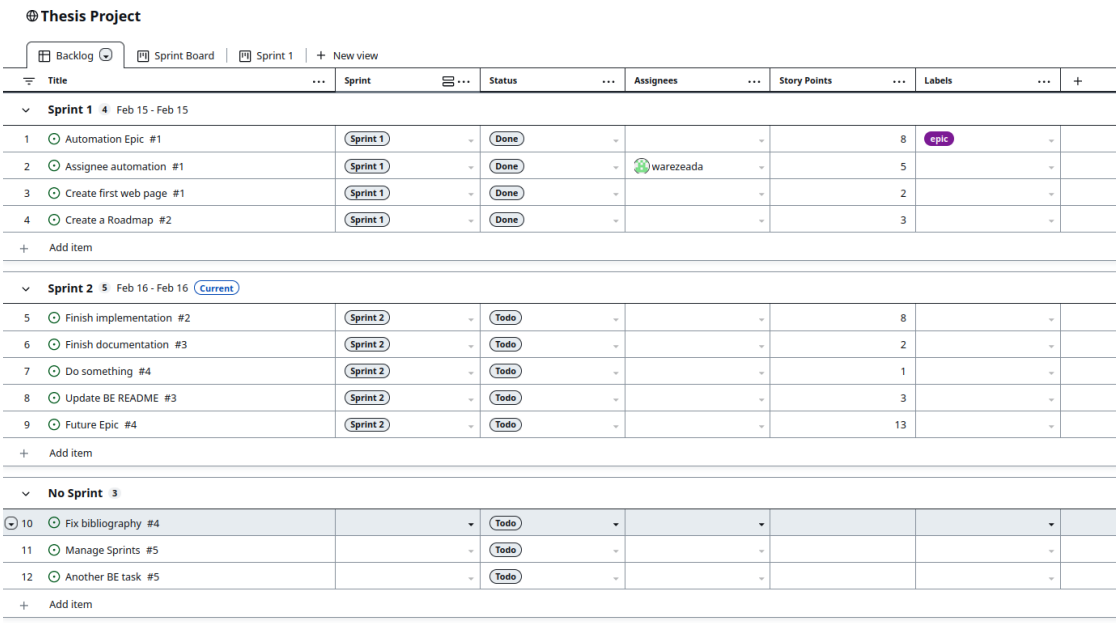
Dlouhodobé plánování začíná tím, že Product Owner vytváří úkoly v Produktového backlogu. Přidáním úkolu vzniká nejdříve pouze draft, náčrt. Ten je potřeba konvertovat na úkol, přičemž se musí vybrat, ke kterému repozitáři se úkol váže. Úkoly zahrnující změny ve více repozitářích jsou přidány k repozitáři *thesis-epics*, je k nim přidán štíte *epic* a je do nich umístěn seznam podúkolů jako tasklist.

V backlogu tým průběžně prochází úkoly, doplňuje odhady, akceptační kritéria a detaily. Odhad ve Story Points lze doplnit přímo z backlogu, popis a akceptační kritéria lze doplnit z otevřeného detailu úkolu.

Plánování Sprintu

Při plánování Sprintu má tým otevřen pohled na backlog, kde přesouvá úkoly z Produktového backlogu do backlogu plánovaného Sprintu. Při tom tým nahlíží do grafu znázorňujícího kolik práce odvedl v předchozích Sprintech.

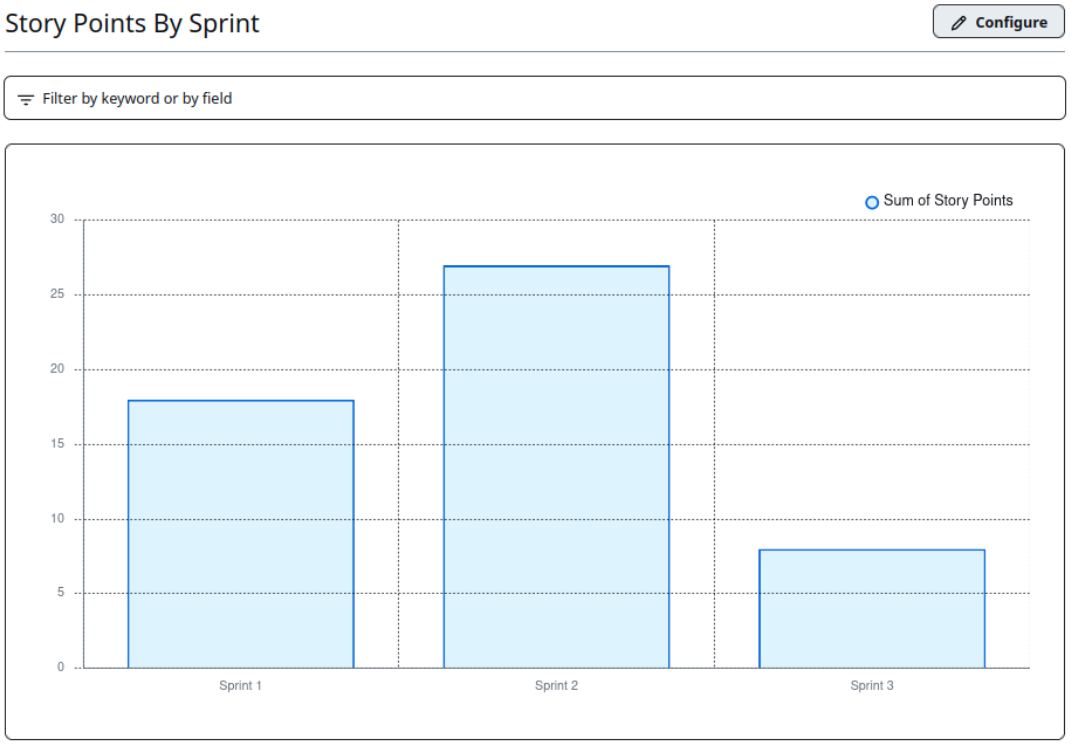
Obrázek 4.7 Plánování v pohledu *Backlog*



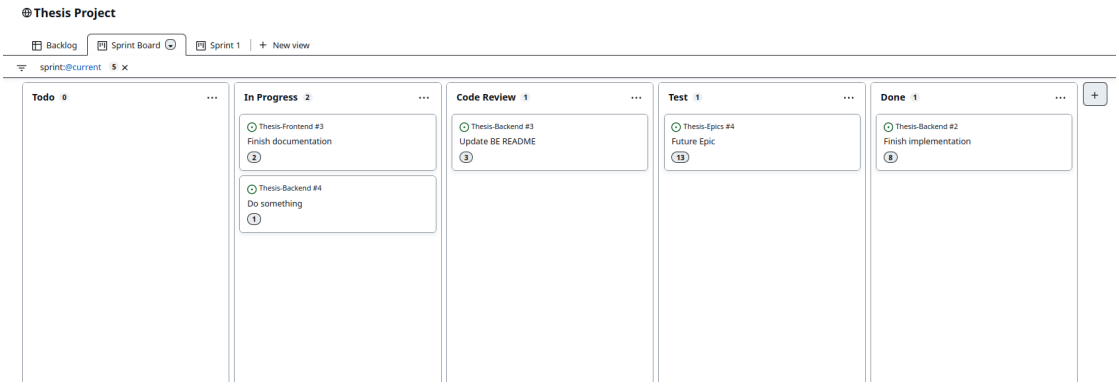
4.3.2 Průběh vývojové iterace

Během vývoje tým udržuje aktuální Sprintovou nástěnku posouváním kartiček s úkoly do sloupců podle stavu úkolu. S tím pomáhá i automatizace po nasazení nových změn, která aktualizuje stav na hodnotu ”Test”značící, že úkol je připraven k otestování. Tým se každodenně schází na standupu, monitoruje průběh Sprintu a snaží se předvídat možná rizika. K tomu by byl vhodný graf typu *Burndown*, ale ten je zatím pouze součástí neveřejné betaverze a nelze tak použít.

Obrázek 4.8 Graf znázorňující počet Story Points na Sprint



Obrázek 4.9 Nástěnka Sprintu v jeho průběhu



4.3.3 Vyhodnocení proběhlé iterace

Na konci iterace tým přesouvá nedokončené úkoly do nového Sprintu. Vhodné by bylo je dát zpět do Produktového backlogu před plánováním nového Sprintu, ale GitHub Projects nepodporuje odstranení Sprintu z úkolu. S pomocí grafu 4.8 může tým porovnat kolik práce dokončeno oproti minulým Sprintům a z toho vycházet při plánování Sprintu nového.

Kapitola 5
Závěr

Hlavním cílem práce bylo analyzovat, otestovat a zhodnotit řízení softwarového projektu v novém nástroji GitHub Projects. Na to došlo v kapitole 4. Před tím ale ještě bylo nutno popsat způsoby agilního řízení projektů v kapitole první, abychom bylo možné lépe analyzovat nástroje a procesy řízení projektů v softwarové firmě. Na základě této analýzy byl vytvořen projekt pomocí nástrojů GitHub Projects a GitHub Issues, ve kterém potom byla provedena simulace řízení iterace projektu.

Ukázalo se, že GitHub projects je ve srovnání s nástrojem Jira, který je použit ve zkoumané firmě (a dle zkušenosti autora i v drtivé většině dalších firem) jednodušší, nabízí méně možností, ale je velmi snadno nastavitelný a má dobrou integraci s dalšími nástroji GitHubu. Nevýhodou je některé důležité funkce, jako je burndown graf, nebo pokročilá automatizace, zatím nebyly vydány pro veřejnost. Pokročilá automatizace jde implementovat vlastními silami pomocí GitHub API a webhooků, ale je to poměrně pracné oproti použití GUI konfigurace automatizace, kterou podporuje Jira Software.

Pokud se již na projektu používá GitHub na hosting repozitáře a na CI/CD, je použití GitHub Projects velmi dobrá volba. Hlavní výhodou oproti nástroji Jira jsou širší možnosti integrace a automatizace procesů v nástroji.

Pokud je ale Git repozitář hostován jinde, a tedy ani CI/CD není implementováno s pomocí platformy GitHub, bylo by vhodnější použít jiný nástroj. Nejspíše nástroj Jira, protože tu dobře zná většina manažerů a vývojářů, a protože nabízí obrovské možnosti, existuje pro ni spousta nástrojů. V případě použití GitLabu pro správu kódu se nabízí použití GitLab Issues, které mají podobné výhody jako GitHub Projects při použitá nástrojů GitHub, jednoduchost a dobrou integraci s ostatními nástroji od GitLabu.

Bibliografie

1. DIGITAL.AI. *State of Agile Report*. 2021. 15. Dostupné také z: https://digital.ai/resource-center/analyst-reports/state-of-agile-report.
2. AGILE ALLIANCE. *Agile Essentials* [online]. 2022. [cit. 2022-04-17]. Dostupné z: https://www.agilealliance.org/agile-essentials/.
3. HAJDIN, Tomáš. *Agilní metodiky vývoje software*. Brno, 2005. Dostupné také z: https://is.muni.cz/th/bi7sx/. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Barbora BÜHNOVÁ.
4. BECK, Kent; BEEDLE, Mike; VAN BENNEKUM, Arie; COCKBURN, Alistair; CUNNINGHAM, Ward; FOWLER, Martin; GRENNING, James; HIGHSMITH, Jim; HUNT, Andrew; JEFFRIES, Ron et al. *Manifesto for Agile Software Development*. 2001. Dostupné také z: http://agilemanifesto.org.
5. KPMG ADVISORY N.V. *Agile Transformation, Survey on Agility*. 2019. Dostupné také z: https://assets.kpmg/content/dam/kpmg/be/pdf/2019/11/agile-transformation.pdf.
6. SUTHERLAND, Jeff; SCHWABER, Ken. *Scrum Guide*. 2020-11. Dostupné také z: https://scrumguides.org/scrum-guide.html.
7. SCHWABER, K. *Agile Project Management with Scrum*. Pearson Education, 2004. Developer Best Practices. ISBN 9780735637900. Dostupné také z: https://books.google.cz/books?id=6pZCAwAAQBAJ.
8. LAKEWORKS. Scrum process. In: *Wikimedia Commons*. 2009. Dostupné také z: https://commons.wikimedia.org/wiki/File:Scrum_process.svg.
9. ŠOCHOVÁ, Zuzana; KUNCE, Eduard. *Agilní metody řízení projektů*. Computer Press, 2019. ISBN 9788025149614. Dostupné také z: https://books.google.cz/books?id=co6jDwAAQBAJ.
10. THE GRAPHQL FOUNDATION. *What is Kanban?* [online]. 2023. [cit. 2023-02-02]. Dostupné z: https://learn.microsoft.com/en-us/devops/plan/what-is-kanban.
11. DAN RADIGAN. *Kanban* [online]. 2023. [cit. 2023-02-02]. Dostupné z: https://www.atlassian.com/agile/kanban.
12. AGILE ALLIANCE. *Extreme Programming* [online]. 2022. [cit. 2022-04-17]. Dostupné z: https://www.agilealliance.org/glossary/xp.
13. *Jira* [online]. 2023. [cit. 2023-02-02]. Dostupné z: https://www.atlassian.com/jira.
14. *Atlassian Company* [online]. 2023. [cit. 2023-02-02]. Dostupné z: https://www.atlassian.com/company.

15. Scrum Board. In: *Jira Software Features* [online]. Atlassian, ©2023 [cit. 2023-02-02]. Dostupné z: https://confluence.atlassian.com/jiracoreserver075/using-active-sprints-976164461.html.
16. LANG, Jean-Phillipe; FELIX, Daniel. *Redmine Wiki* [online]. 2023. [cit. 2023-02-12]. Dostupné z: https://www.redmine.org/projects/redmine/wiki.
17. LANG, Jean-Phillipe; FELIX, Daniel. *Redmine Installation* [online]. 2023. [cit. 2023-02-12]. Dostupné z: https://www.redmine.org/projects/redmine/wiki/RedmineInstall.
18. Redmine Issue List. In: [online]. 2023 Jean-Phillipe Lang, ©2006-2014 [cit. 2023-02-02]. Dostupné z: https://www.redmine.org/projects/redmine/issues.
19. GITLAB. *About GitLab* [online]. 2023. [cit. 2023-02-09]. Dostupné z: https://about.gitlab.com.
20. GITLAB. *Plan and track work documentation* [online]. 2023. [cit. 2023-02-09]. Dostupné z: https://docs.gitlab.com/ee/topics/plan_and_track.html.
21. GitLab Issue Board. In: *GitLab Documentation* [online]. 2023 GitLab B.V., ©2023 [cit. 2023-02-02]. Dostupné z: https://docs.gitlab.com/ee/user/project/img/issue_boards_core_v14_1.png.
22. GITHUB. *GitHub Features* [online]. 2022. [cit. 2022-12-12]. Dostupné z: https://github.com/features.
23. GITHUB. *GitHub Enterprise Documentation* [online]. 2022. [cit. 2022-12-12]. Dostupné z: https://docs.github.com/en/enterprise-cloud@latest/admin/overview/about-github-for-enterprises.
24. GITHUB. *GitHub Projects Documentation* [online]. 2022. [cit. 2022-12-12]. Dostupné z: https://docs.github.com/en/issues/planning-and-tracking-with-projects.
25. GITHUB. *GitHub Pricing* [online]. 2022. [cit. 2022-12-18]. Dostupné z: https://github.com/pricing.
26. GitHub Projects Table. In: *GitHub Projects Documentation* [online]. GitHub, ©2022 [cit. 2022-04-18]. Dostupné z: https://docs.github.com/assets/cb-160780/images/help/issues/projects_table.png.
27. *Jira Software Resources* [online]. 2023. [cit. 2023-02-02]. Dostupné z: https://www.atlassian.com/company.
28. GitHub Projects Select Template. In: *GitHub Projects Documentation* [online]. GitHub, © 2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/assets/cb-46006/images/help/issues/projects-select-template.png.
29. GITHUB. Projects Views. In: *GitHub Documentation* [online]. 2022 GitHub, ©2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/en/issues/planning-and-tracking-with-projects/customizing-views-in-your-project.
30. GitHub Projects Board View. In: *GitHub Projects Documentation* [online]. GitHub, ©2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/assets/cb-194569/images/help/projects-v2/example-board.png.
31. GitHub Projects Roadmap View. In: *GitHub Projects Documentation* [online]. GitHub, © 2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/assets/cb-114760/images/help/projects-v2/example-roadmap.png.
32. GITHUB. Projects Automation. In: *GitHub Documentation* [online]. 2022 GitHub, ©2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/en/issues/planning-and-tracking-with-projects/automating-your-project.
33. THE GRAPHQL FOUNDATION. *Introduction to GraphQL* [online]. 2023. [cit. 2023-01-01]. Dostupné z: https://graphql.org/learn/.

34. GITHUB. Webhooks. In: *GitHub Documentation* [online]. 2022 GitHub, ©2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/en/developers/webhooks-and-events/webhooks.

35. GITHUB. Actions. In: *GitHub Documentation* [online]. 2022 GitHub, ©2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

36. GITHUB. Projects Insights. In: *GitHub Documentation* [online]. 2022 GitHub, ©2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/en/issues/planning-and-tracking-with-projects/viewing-insights-from-your-project.

37. GitHub Projects Insights Chart. In: *GitHub Projects Documentation* [online]. GitHub, © 2022 [cit. 2022-12-12]. Dostupné z: https://docs.github.com/assets/cb-35124/images/help/issues/column-chart-example.png.

38. GITHUB. Projects Best Practices. In: *GitHub Documentation* [online]. 2022 GitHub, ©2022 [cit. 2022-12-14]. Dostupné z: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/best-practices-for-projects.

39. GitHub Task List Rendered. In: *GitHub Documentation* [online]. GitHub, ©2022 [cit. 2022-12-14]. Dostupné z: https://docs.github.com/assets/cb-218613/images/help/writing/task-list-rendered.png.

40. Sprint Board. In: *Jira Software Support* [online]. Atlassian, ©2022 [cit. 2022-12-20]. Dostupné z: https://confluence.atlassian.com/jiracoreserver075/using-active-sprints-976164461.html.

41. Scrum Backlog. In: *Jira Software Support* [online]. Atlassian, ©2022 [cit. 2022-12-20]. Dostupné z: https://confluence.atlassian.com/jirasoftwareserver/creating-your-backlog-938845071.html.

42. GITHUB. Issues. In: *GitHub Documentation* [online]. 2022 GitHub, ©2022 [cit. 2022-12-14]. Dostupné z: https://docs.github.com/en/issues/tracking-your-work-with-issues.

Obsah přiloženého média

- `exe` — adresář se spustitelnou formou implementace
- `src`
  - `impl` — zdrojové kódy implementace
  - `thesis` — zdrojová forma práce ve formátu LaTeX
- `text` — text práce
  - `thesis.pdf` — text práce ve formátu PDF