EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103003/I/2020/36084588817095172

SOFTVÉROVÉ NÁSTROJE PROJEKTOVÉHO RIADENIA

PLÁNOVANIE PROJEKTOVÉHO PORTFÓLIA

Diplomová práca

2020 **Bc. Ivana Lieskovská Drábiková**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

SOFTVÉROVÉ NÁSTROJE PROJEKTOVÉHO RIADENIA

PLÁNOVANIE PROJEKTOVÉHO PORTFÓLIA

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Ekonómia a manažment

Školiace pracovisko: Katedra operačného výskumu a ekonometrie FHI

Vedúci záverečnej práce: Ing. Pavel Gežík, PhD.

Bratislava 2020 **Bc. Ivana Lieskovská Drábiková**

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracovala samostatne a že som uviedla všetku použitú literatúru.

Dátum: **.....**

Meno a priezvisko

Poďakovanie

Touto cestou by som chcela poďakovať vedúcemu práce a svojej rodine.

Abstrakt

LIESKOVSKÁ DRÁBIKOVÁ, Ivana: *Softvérové nástroje projektového riadenia*. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra operačného výskumu a ekonometrie. – Vedúci záverečnej práce: Ing. Pavel Gežík, PhD. – Bratislava: FHI EU, 2020, 65 s.

Cieľom tejto diplomovej práce je porovnať existujúce nástroje plánovania projektového riadenia a následne aplikovať vybraný nástroj na úlohu plánovania projektového portfólia. Práca je rozdelená na 3 hlavné kapitoly. Prvá kapitola popisuje súčasný stav situácie a teoretické poznatky o metódach plánovania projektov. Druhá kapitola popisuje ciele práce a použité metódy. V tretej kapitole sú popísané jednotlivé výstupy porovnania skúmaných nástrojov, výsledky aplikácie vybraného nástroja a poukázanie na jednotlivé odlišnosti, výhody a nevýhody skúmaných nástrojov. Výsledok diplomovej práce slúži ako riešenie reálneho plánovacieho problému v prostredí agilného softvérového vývoja mobilnej aplikácie a pomáha naplánovať prácu na komponentoch projektového portfólia s ohľadom na minimalizáciu časového rozpätia či oneskorenia projektov.

Kľúčové slová:

plánovanie, portfólio projektový manažment, job schop scheduling, OptaPlanner, agilnný vývoj

Abstract

LIESKOVSKÁ DRÁBIKOVÁ, Ivana: *Software Tools for Project Management*. – University of Economics in Bratislava. Faculty of Economic Informatics; Department of Operations Research and Econometrics. – Advisor: Ing. Pavel Gežík, PhD. – Bratislava: FHI EU, 2020, 65 pp.

The aim of this thesis is to compare the existing tools for project management planning and then apply the selected tool to the problem of project portfolio planning. The thesis is divided into three main chapters. The first chapter describes the current state of the situation and theoretical knowledge about project planning methods. The second chapter describes the objectives of the work and the methods used. The third chapter describes the individual outputs of the comparison of the examined tools, the results of the application of the selected tool and pointing out the individual differences, advantages and disadvantages of the examined tools. The result of the diploma thesis serves as a solution to a real planning problem in the environment of agile software development of a mobile application and helps to plan work on components of a project portfolio with regard to minimizing the time span or delays of projects.

Key words:

planning, portfolio project management, job shop scheduling, OptaPlanner, agile development

Obsah

Zoznam ilustrácií

Zoznam tabuliek

Zoznam použitých skratiek

BE – Back End

DB - Databáza

JVM – Java Virtual Machine

JSSP – Job Shop Scheduling Problem

LeSS – Lare Scale Scrum

LP – Linear programming

MIP – Mixed integer programming

MILP – Mixed iteger linear programming

OR – Operational research

SDK – Software Development Kit

SP – Story Point

SP/ *i* – Story Point per interation

VHR, ENR, CLN, REST, CABS, RBC, MSA – skratky projektov

XML – Extensible markup language

Úvod

Existuje dlhý zoznam metodík, ktoré sa venujú projektovému manažmentu. Každá môže používať iné metódy na riešenie výziev, ktoré projektový manažment obnáša. Jednou z týchto výziev je plánovanie projektu a s ním súvisiace optimalizačné úlohy, na ktoré sa najčastejšie využívajú nástroje lineárneho programovania.

V tejto diplomovej práci sa nebudeme zaoberať jednotlivými metódami plánovania či porovnávaním metodík projektového manažmentu, o ktorých už bolo publikovaných mnoho odborných textov. Posunieme sa ešte o krok ďalej, a to k problému plánovania projektového portfólia.

Pri plánovaní projektu potrebujeme rátať s plánovaním zdrojov vzhľadom na obmedzený čas a určené množstvo práce. Plánovanie portfólia je viacdimenzionálna úloha a treba vziať do úvahy možné odlišnosti každého projektu (napr. rozdielne metodiky), ako aj ich spoločné atribúty (napr. zdieľané zdroje). Vo vedení softvérových firiem sa do popredia dostávajú agilné metodiky vývoja a preberajú miesto tradičnému vodopádovému prístupu. Ak je agilnými metodikami vedené veľké množstvo projektov, ktoré spadajú do projektového portfólia, nastáva komplikovaná situácia, ktorá sa týka napríklad využitia zdrojov alebo odhadu ukončenia každého projektu v portfóliu. Vynárajú sa otázky, ako efektívne plánovať portfólio agilných projektov? Ako efektívne zdieľať kapacity medzi jednotlivými tímami? Kedy môžeme počítať s ukončením každého projektu?

V práci sa venujeme práve týmto otázkam, konkrétne ako odhadnúť a naplánovať objem práce, ktorú dokážu developerské tímy dokončiť za jednu iteráciu a ako optimálne plánovať portfólio projektov vo veľkých a agilných tímoch za dodržania pevne stanovených míľnikov. Taktiež sa zameriava na tému realistického časového odhadu dokončenia plánovaného vývoja v portfólio manažmente, čo je náročná úloha, pri ktorej treba pracovať s veľkou mierou neurčitosti. Cieľom práce je teda preštudovať a porovnať nástroje, ktoré plánovanie portfólia agilne riadených projektov dokážu sprehľadniť a podporiť na úrovni rozhodovacích procesov. Vybraný nástroj následne budeme aplikovať na konkrétny problém plánovania z praxe.

Poznatky využité v tejto diplomovej práci vychádzajú z praktických skúsenosti autorky na pozícii portfólio manažérky v automobilovej spoločnosti, z oddelenia so zameraním na vývoj mobilnej aplikácie a diplomová práca samotná pomáha riešiť reálny problém plánovania projektového portfólia.

Teoretické znalosti z oblasti riadenia portfólia projektov sú čerpané najmä z anglickej knihy *The Standard for Portfolio Management*, ktorá je veľmi stručnou a všeobecnou príručkou o projektovom portfólio manažmente. Čo sa týka agilnej metodiky, ako zdroj informácií slúžili konzultácie s agilným koučom na spomínaných projektoch, vzdelávacie kurzy a niekoľko príručiek dotýkajúcich sa tém agilného vývoja, metodiky Scrum a LeSS.

Prvá časť práce sa venuje najmä teoretickým poznatkom, vysvetleniu pojmu projektu, projektového portfólia a portfólio manažmentu. Popisuje súčasný stav situácie a teoretické poznatky o metódach plánovania projektov a portfólií projektov.

Druhá kapitola popisuje ciele práce a použité metódy. Je venovaná popisu optimalizačných nástrojov slúžiacich na riešenie plánovacích úloh. V tejto práci je vybraných niekoľko nástrojov, ktoré sú skúmané do rôznej hĺbky a ktoré môžu byť aplikované na problém plánovania projektového portfólia v agilnom prostredí.

V tretej kapitole sú popísané jednotlivé výstupy porovnania skúmaných nástrojov, výsledky aplikácie vybraného nástroja a poukázanie na jednotlivé odlišnosti, výhody a nevýhody skúmaných nástrojov. Kapitola tiež bližšie popisuje možnosti využitia spomínaných nástrojov pri konkrétnych typoch problémov.

Výsledok diplomovej práce slúži ako riešenie reálneho plánovacieho problému v prostredí agilného softvérového vývoja mobilnej aplikácie a pomáha naplánovať prácu na komponentoch projektového portfólia s ohľadom na minimalizáciu časového rozpätia a oneskorenia projektov.

1 Súčasný stav riešenej problematiky doma a v zahraničí

Ťažiskom tejto kapitoly je popis teoretických poznatkov z oblasti riadenia projektov. Metodika ako pojem je charakterizovaná ako súbor doporučených praktík a postupov, ktoré sprevádzajú projekt počas celého jeho životného cyklu. Projektová metodika spája predstavy zainteresovaných strán a zjednocuje komunikačný jazyk o zahájení, priebehu a ukončení projektu, preto považujeme za vhodné popísať v tejto kapitole vybrané metodiky. Okrem toho táto kapitola popisuje známe poznatky o projektoch, projektovom riadení či vysvetľuje pojem projektové portfólio.

Kapitola ďalej stručne zhŕňa rozdiely medzi vodopádovým a iteratívnym prístupom a vysvetľuje termíny používané v praktickej časti diplomovej práce. Zameriava sa na popis softvérových nástrojov pre plánovanie a plánovanie vývoja v agilnom prostredí. Spomíname tu softvérové nástroje, ktoré pomáhajú plánovať a optimalizovať proces riadenia projektov a uľahčujú organizáciu práce projektového manažéra.

V kapitole tiež detailnejšie popisujeme použitie konkrétnych pojmov, ktoré využívame v praktickej časti práce. Ide napríklad o použitie relatívnych jednotiek (*Story points*), ktoré sa používajú v metodike Scrum a slúžia na odhadovanie náročnosti úloh pri vývoji softvéru. Okrem toho vysvetľujeme pojem výkonnosti tímu (*velocity*) a jeho výpočet pomocou relatívnych jednotiek.

1.1 Projekt, portfólio

Táto podkapitola stručne definuje pojem projekt, portfólio, projektové riadenie a vysvetľuje vzťahy medzi projektom, programom portfóliom.

1.1.1 Projekt

Projekt je definovaný ako časovo ohraničené úsilie, ktoré je zamerané na vznik určitej služby alebo produktu. Každý projekt má životný cyklus, má definovaný začiatok a koniec, no po ukončení projektu by mal mať výsledok projektu trvalý prínos.

Projekt by mal byť jedinečný a mal by riešiť existujúci problém pomocou uceleného súboru činností.

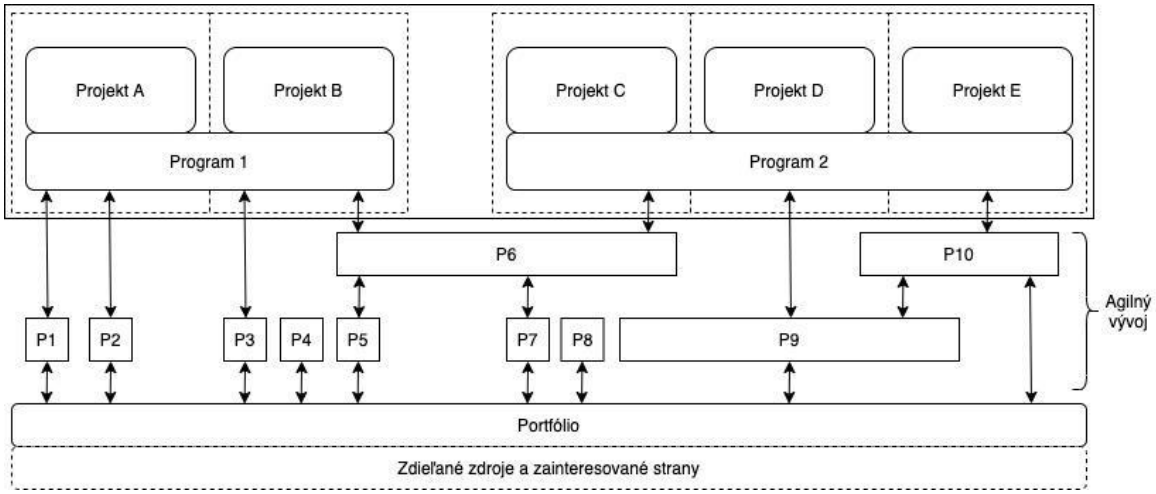
Jedna z často používaných metodík, PRINCE2, pojem projekt vysvetľuje nasledovne: „*Projekt je dočasná organizácia vytvorená za účelom poskytnutia jedného alebo viacerých produktov na základe schváleného obchodného zámeru.*"[1] Pod pojmom organizácia sa rozumie projektový tím ľudí, ktorí sa podieľajú na projekte. Obchodný zámer je dokument, ktorý obsahuje informácie o potrebách realizácii projektu z obchodného hľadiska a potrebné zdroje na realizáciu daného projektu.

1.1.2 Portfólio

Portfólio je súbor projektov, programov, pridružených portfólií a operácií, ktoré sú riadené ako celok s účelom dosiahnuť strategické ciele. Jednotlivé komponenty môžu byť na sebe závislé, nemusia spolu súvisieť a sú kvantifikovateľné (napr. prioritizácia). Rovnako ako projekt, aj portfólio má svoj životný cyklus, no na rozdiel od projektu je životnosť portfólia dlhšia a má väčšiu pozornosť manažmentu. Komponenty portfólia sa delia o zdieľané zdroje alebo ich časť. Zánik portfólia môže nastať vtedy, keď už portfólio nemá pre danú organizáciu význam, prípadne sa združí s iným portfóliom alebo rozdelí na ďalšie portfólia. Portfólio je väčšinou súčasťou hierarchickej štruktúry, no môže byť aj súčasťou iného, väčšieho portfólia. Portfólio by malo vždy smerovať k organizačným strategickým cieľom. [5] Typickým príkladom portfólia je napr. produktová rada (napr. vozidla) alebo IT portfólio.

[1] OGC. Managing successful projects with PRINCE2. 5. vydanie. Londýn: TSO, 2009. s. 3-4. ISBN 978-011-3310-593.

Obrázok 1: Portfólio [2]



1.1.3 Vzťah medzi projektom, programom a portfóliom

Vzťah medzi portfóliami, programami a projektmi je taký, že portfólio sa vzťahuje na súbor projektov, programov, subportfólií a operácií riadených ako skupina na dosiahnutie strategických cieľov. Programy sú zoskupené do portfólia a pozostávajú z podprogramov, projektov alebo inej práce, ktorá je riadená koordinovane na podporu portfólia. Jednotlivé projekty, ktoré sú buď v rámci programu alebo mimo neho, sa stále považujú za súčasť portfólia. Aj keď projekty alebo programy v rámci portfólia nemusia byť vzájomne prepojené, sú spojené so strategickým plánom organizácie prostredníctvom portfólia organizácie. [5]

1.1.4 Projektové riadenie

Projektový manažment je definovaný ako „*aplikácia vedomostí, zručností, nástrojov a techník na projektové aktivity za cieľom dosiahnutia projektových požiadaviek*“.[3] Zároveň ide o koordináciu aktivít, ktoré súvisia s riadením ľudských,

[2] Zdroj: The Standard for Portfolio Management. 4. vydanie. Pennsylvania: Project Management Institute, 2017. s 4. ISBN 978-162825-197-5.

[3] Project Management Institut. A Guide To The Project Management Body Of Knowledge. 5 vydanie. Project Management Institut, 2013. 589 s. ISBN 978-1935589679.

finančných a materiálnych zdrojov. Sledovanie projektových cieľov musí byť v súlade so stratégiou podniku a rešpektovať stanovené metodiky.

1.1.5 Riadenie portfólia

Riadenie portfólia je centralizovaný manažment komponentov portfólia. Komponenty portfólia pritom zdieľajú zdroje, ktoré do nich organizácia investuje a cieľom portfólio manažéra je riadiť tieto komponenty s ohľadom na stanovené strategické ciele - monitorovaním, ohodnocovaním, integráciou, prioritizáciou, optimalizáciou, vyvažovaním, riadením procesu tranzície, kontrolovaním, zahajovaním a ukončovaním komponentov portfólia.

Primárny cieľ spojenia portfólio manažmentu a organizačnej stratégie je nastaviť vyvážený a realistický plán, ktorý pomôže organizácii dosiahnuť svoj strategický cieľ. To je možné pomocou riadenia šiestich domén [6]:

- riadenie portfólia,
- riadenie životného cyklu portfólia,
- riadenie rizík portfólia,
- riadenie kapacít portfólia,
- riadenie hodnôt portfólia,
- zapájanie zainteresovaných strán.

1.2 Metodiky

V tejto kapitole predstavíme najčastejšie používané metodiky projektového riadenia, ktoré sa využívajú pri vývoji softvéru. Pôjde o vodopádový model a agilné metodiky. Kapitola ďalej stručne zhŕňa rozdiely medzi vodopádovým a iteratívnym prístupom a vysvetľuje termíny používané v praktickej časti diplomovej práce.

1.2.1 *Vodopádový model vývoja*

Vodopádový model je klasickým prístupom a funguje na báze sekvenčného spracovania jednotlivých fáz životného cyklu projektu. Nejedná sa o metodiku, pretože nepopisuje konkrétne postupy, popisuje iba fázy vývoja. Tento model bol predstavený v roku 1970 Winstonom W. Roycom ako paradoxne nesprávny prístup k vývoju, no napriek tomu je široko využívaný.

Názov modelu vznikol až neskôr, a to kvôli podobnosti so zvažujúcim sa tokom, kedy činnosti prechádzajú jednotlivými fázami až k výslednému projektu. Fázy tohto modelu sú nasledovné [7]:

1. zber a spracovanie systémových a softvérových požiadaviek,
2. analýza, jej výsledkom sú modely a schémy,
3. dizajn - návrh architektúry,
4. implementácia - fáza programovania a integrácie softvéru,
5. testovanie,
6. operácie - údržba vrátane inštalácie, migrácie a podpory celého systému.

1.2.2 *Iteratívny (inkrementálny) model*

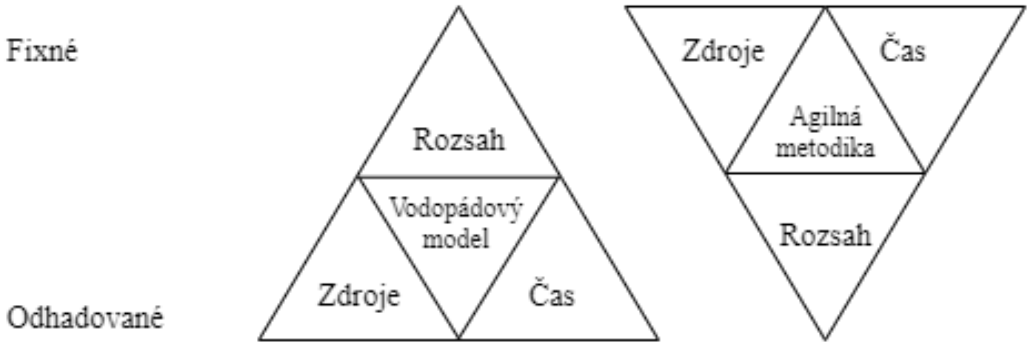
Inkrementálny vývoj softvéru je často používaný pri vývoji softvérových produktov. Hlavný rozdiel, v čom sa iteratívny prístup odlišuje od vodopádového vývoja, spočíva v tom, že namiesto dodania monolitického systému po relatívne dlhom čase vývoja sa menšie časti funkcionalít implementujú postupne. Tento prístup má oproti tradičnému vodopádovému modelu veľa výhod.

Po prvé, požiadavky môžu byť uprednostňované tak, aby sa tie najdôležitejšie splnili ako prvé a stali sa súčasťou nového systému skôr. V dôsledku toho zostávajú menej dôležité požiadavky odkladané až neskôr, a v prípade nedostatku času alebo zdrojov sú najmenej potrebné požiadavky z vývoja vynechané. Po druhé to znamená, že zákazníci dostávajú čiastočné dodávky priebežne, a preto je pravdepodobnejšie, že budú vedieť včas poskytnúť spätnú väzbu.

Po tretie, keďže je časový úsek jednej iterácie menší, je jednoduchšie naplánovať a odhadnúť, koľko práce sa skutočne stihne dokončiť v danom časovom intervale a za aké zdroje. Po štvrté, spätnú väzbu od používateľov je možné získať v každej fáze a podľa nej prispôsobiť či prioritizovať následný vývoj. Po piate, inkrementálny prístup umožňuje oveľa lepšiu reakciu na zmeny alebo doplnenia požiadaviek.

Spomínané výhody iteratívneho prístupu sú vyžívané v tzv. agilnom vývoji. [2] Agilných metodík samozrejme existuje niekoľko. Niektoré sa zameriavajú na činnosti (extrémne programovanie, pragmatické programovanie, agilné modelovanie), ďalšie na riadenie toku práce (Scrum, Kanban).

Obrázok 2: Projektový trojimperatív [4]



Hlavný rozdiel medzi vodopádovým prístupom a agilným prístupom s ohľadom na *triple constraint*, projektový trojimperatív, najlepšie ilustruje obrázok vyššie. Trojimperatív popisuje vzťah hlavných parametrov projektu: čas, zdroje a rozsah projektu. V tradične riadenom projekte je rozsah pevne stanovený na začiatku projektu. Ďalšie dva parametre sú flexibilnejšie, resp. je možné ich v čase mierne upravovať.

V agilnom prístupe je trojuholník vzťahu čas - zdroje - rozsah obrátený. Zdroje (tímy) a čas (časovo ohraničené iterácie) sú pevne stanovené a flexibilný je dodaný rozsah projektu, pričom ako prvé sú dodávané tie časti produktu (*deliverables*), ktoré majú pre zákazníka najvyššiu hodnotu.

[4] Zdroj: Role of The Agile Project Manager. [online]. Dostupné na internete: < https://www.virtualprojectconsulting.com/role-of-the-agile-project-manager/ >

1.2.3 Scrum

Scrum je konkrétna agilná metodika, ktorá popisuje presné postupy a najlepšie praktiky (*best-practices*). Využíva iteratívny model, pričom jedna iterácia spravidla trvá 1-4 týždne. Role v projektovom tíme sú trochu odlišné od bežného projektového tímu. Uvádzame niekoľko dôležitých pojmov, ktoré sú nevyhnutné pre pochopenie fungovanie tejto metodiky.

Sprint je jedna iterácia, ktorá trvá spravidla 1-4 týždne. *Sprint* má daný jasný rozvrh. Každodenne sa organizujú krátke synchronizačné stretnutia, tzv. *Daily Scrum* resp. *Stand-up*, počas ktorých každý člen tímu stručne zhrnie, čo sa mu podarilo dokončiť včera, na čom má v pláne pracovať dnes a aké vidí problémy resp. riziká. Celkovo *Stand-up* trvá asi 15 minút.

Product Backlog je zoznam úloh a funkcionalít, ktoré si zákazník želá. Produktový *backlog* je prioritizovaný zoznam úloh, ktoré majú byť hotové, je to zoznam požiadaviek produktu. Z Produktového *backlogu* sa vyberá podmnožina úloh, ktoré majú byť dokončené v nasledovnom *sprinte*. [2]

Sprint Planning je stretnutie, ktoré sa koná na začiatku každého *sprintu*. Cieľom stretnutia je stanoviť si cieľ nasledujúceho *sprintu*, prediskutovať ďalšie kroky a z produktového *backlogu* ich zaradiť do *sprint backlogu*. *Sprint planning* pozostáva z Retrospektívy, počas ktorej tím zhodnotí pozitíva a negatíva minulého *sprintu* a dohodne sa na akčných krokoch, ktoré zabezpečia zlepšenie. Ďalej nasleduje *Sprint Review*, kedy sa prezentujú výsledky získané za minulý *sprint* a následne sa prechádza k fáze plánovania.

Product Owner je jedna z rolí v scrumovom tíme. *Product Owner* alebo vlastník produktu predstavuje hlas biznisu. Rozhoduje o tom, ktorých úlohám z produktového *backlogu* určí prioritu pre vývoj v najbližších *sprintoch*. Je tzv. “úzkym hrdlom” pri zaraďovaní funkcionalít požadovaných zákazníkom do vývoja.

Scrum Master je “majster” scrumu a pôsobí ako prostredník medzi tímom a vonkajšími negatívnymi vplyvmi. Nie je to tradičný projektový manažér alebo tímový líder. Okrem toho sa stará o dodržiavanie metodiky, zaisťuje scrumové ceremónie a stará sa o zabezpečenie funkčnej komunikácie a fungovanie celého tímu.

Scrumový tím sa stretáva pravidelne na rôznych ceremóniách, ktoré zahŕňajú krátke každodenné synchronizačné schôdzky (*Daily scrum*). Vývojový tím je multifunkčný a samoorganizujúci. Tvorí ho spravidla 3-9 členov, vrátane vývojárov, testerov, analytikov, ktorí spoločne pracujú na vytvorení doručiteľného inkrementu. [2]

1.2.4 LeSS

Large Scale Scrum je adaptácia metodiky scrum a považujeme za dôležité spomenúť ju v tejto kapitole, pretože táto metodika je využívaná pri riadení portfólia projektov, ktoré používame v prípadovej štúdii. LeSS je framework, ktorý využíva Scrum metodiku, vo veľkoformátovom prostredí - čiže pri riadení viacerých scrumových tímov. [13] Prvý level LeSS je dizajnovaný na maximálne 8 tímov, druhý level známy ako "*LeSS Huge*" pridáva elementy pre tímy, ktoré majú niekoľko stoviek vývojárov.

V LeSSe pribúdajú nové role, ktoré podporujú koncept scrumovej metodiky. Sú to napríklad agilní kouči, ktoré pomáhajú zabezpečiť celkový chod tímov a dodržiavanie metodiky. Navyše okrem *Product Ownera* existuje aj rola tzv. *Area Product Owner*, ktorý má na starosti menšie časti vyvíjaného produktu.

1.2.5 Kanban

Kanban je japonská metóda používaná na riadenie práce. Slovo Kanban je odvodené z dvoch japonských slov a prekladá sa ako karta resp. tabuľa. Metóda bola vyvinutá firmou Toyota, ktorá v štyridsiatych rokoch dvadsiateho storočia študovala systém riadenia supermarketov. [2] V súčasnosti sa metóda používa aj pri vývoji softvéru. Požiadavky existujú vo forme kariet a každá karta sa nachádza v určitom stave procesu vývoja, takže karty zároveň signalizujú priebeh vývoja

1.3 Nástroje na plánovanie vývoja softwaru

V tejto kapitole sa venujeme popisu nástrojov, ktoré sú často používané pri plánovaní, či už vývoja alebo výrobného procesu všeobecne.

Kapitola je rozdelená na dve časti. Prvá časť sa venuje najčastejším druhom optimalizačných problémov, s ktorými sa stretávame pri plánovaní. Ide o úlohy a metódy operačného výskumu a matematického programovania. Druhá časť je zameraná na softvérové nástroje, ktoré pomáhajú a podporujú manažérske rozhodnutia.

Kapitola bude používať mnoho anglických pojmov, ktoré sa bežne vyskytujú v literatúre a do slovenčiny ich nie je možné uspokojivo preložiť, prípadne ide o zaužívané názvy, ktoré sa používajú v praxi. Ide najmä o názvy optimalizačných úloh alebo súčasti nástrojov či terminológia súvisiaca so spomínanými metodikami a nástrojmi.

Keďže čitateľ ich význam spozná skôr pod pôvodným názvom ako pod slovenským prekladom, budeme používať anglické názvy, no budeme sa snažiť dostatočne zrozumiteľne popisovať daný problém aj v slovenčine.

1.1.6 Najčastejšie druhy problémov a úloh v procese plánovania

Critical Path Method je najpoužívanejšou metódou hľadania kritickej cesty v sieti. „*Vychádza z časovej analýzy údajov spojených s riadením projektu, resp. jeho realizáciou. Jedná sa najmä o termíny realizácie jednotlivých činností, pričom sa zameriava na určovanie začiatkov a koncov. Najdôležitejší je pritom koniec poslednej činnosti, ktorí hovorí o termíne ukončenia projektu.*“[5] V projektovom manažmente je to často používaný nástroj. Akékoľvek zdržanie projektu v ktorejkoľvek jeho fáze tak má za následok, že bude oneskorený celý projekt. Na vyjadrenie kritickej cesty je potrebné najskôr poznať poradie úloh, v akom majú po sebe nasledovať. Tieto úlohy je potom možné zobraziť na Ganttovom diagrame, ktorý je často používaným pri znázornení aktivít v projektovom riadení na časovej osi.

NP-ťažké úlohy/ NP-úplné úlohy sú úlohy, ktoré sú vypočítateľné v nedeterministickom polynomiálnom čase. Pre také úlohy je charakteristické, že je relatívne rýchle overiť správnosť riešenia, no nájdenie riešenia trvá veľmi dlho. NP-úplné úlohy sú preto často riešené heuristikami a aproximačnými algoritmami.

---

[5] BREZINA, Ivan – ČIČKOVÁ, Zuzana – GEŽÍK, Pavel. Sieťová analýza. Bratislava: Vydavateľstvo EKONÓM, 2012. 119 s. ISBN 978-80-225-3503-8.

Príklady NP-ťažkých úloh môžeme nájsť napríklad v kryptografii, či v teórii kódov. Medzi najznámejšie NP-úplné úlohy patrí napríklad úloha o batohu, problém obchodného cestujúceho, zafarbovanie grafu, hľadanie najkratšej hamiltonovskej kružnice, či ďalšie rozvrhovacie úlohy. Pri riešení tohto typu úloh môžeme použiť exaktný algoritmus pre úlohy malých rozmerov.

Pre riešenie úloh veľkých rozmerov sú používané heuristické algoritmy a hovoríme o hľadaní najlepšieho možného resp. suboptimálneho riešenia. [1] Nemôžeme si totiž byť istí, že dané riešenie bude aj optimálne.

Hľadanie najlepších možných riešení NP-ťažkých a NP-úplných úloh v zadanom čase je možné pomocou výpočtových nástrojov, ktoré budeme spomínať v podkapitole nižšie.

Priraďovacie úlohy priraďujú zdroje požiadavkám. Základná predstava o tejto úlohe, ktorá je predstavovaná v literatúre [3], popisuje problém dispečera taxislužby, ktorý dostal *n* objednávok od zákazníkov a na parkovisku stojí *n* taxíkov. Úlohou dispečera je vykonať *n* bivalentných rozhodnutí, a to, či taxík priradí alebo nepriradí danému zákazníkovi. Cieľom úlohy je minimalizovať súčet dĺžok ciest všetkých taxíkov.

Problém priraďovania úloh alebo *Task assigning problem* je modifikáciou priraďovacej úlohy s pridanými podmienkami. V tejto úlohe priraďujeme zamestnancom úlohy podľa ich príbuznosti k zákazníkovi, ktorý je zadávateľom danej úlohy. [18] Na základe tejto príbuznosti sa potom vypočíta trvanie spracovania danej úlohy.

Rozvrhovacie úlohy (*Scheduling problems*) sú úlohy, v ktorých na rozdiel od priraďovacej úlohy priraďujeme zdrojom s rôznou výkonnosťou úlohy v čase a poradí. Cieľom je minimalizovať časové rozpätie, ktoré tvorí súčet spracovania daných úloh.

Job shop scheduling problem je typ optimalizačnej úlohy, v ktorej je úloha priradená každému zdroju v určitý čas. Základný model popisuje problém úloh, ktoré majú byť priradené strojom s rôznou výkonnosťou spracovania. Cieľom úlohy je minimalizovať časové trvanie rozvrhu, pričom nezáleží na poradí, v akom sú úlohy spracované. [19]

Flow shop scheduling problem je typ úlohy, ktorý je často používaný vo výrobnom prostredí, ktoré produkuje veľký objem produktov, napríklad. Automobilové závody. Prechod medzi jednotlivými fázami je pevne stanovený a vždy rovnaký [12] (napr. lisovňa, zvarovňa, lakovňa). (Produkt sa vráti do predošlej fázy len vo výnimočných prípadoch, keď bol na ňom nájdený defekt a podobne.) Každý produkt prechádza pracovaním jednotlivými zariadeniami sériovo, vždy jedenkrát, pričom záleží na poradí operácií. Snahou úlohy je minimalizovať čas nečinnosti zariadení a minimalizovať čas čakania. Tento typ problému je špeciálnym prípadom predošlej úlohy (*Job shop scheduling*).

Problém rozvrhovania zamestnancov je označovaný aj ako *Nurse scheduling problem* resp. *Nurse rostering*. Ide o model úlohy, v ktorom sú na smeny priradzované zdravotné sestry s ohľadom na požiadavky, ako napríklad počet odpracovaných hodín, preferencie pracovných dní, počet po sebe nasledujúcich pracovných dní, striedanie smien atď.

Problém plánovania projektov je známy aj ako *Project job scheduling problem*. cieľom úlohy je rozvrhnúť všetky úlohy v čase tak, aby sa minimalizovalo oneskorenie projektu. [14] Každá úloha je súčasťou projektu a vyžaduje iný spôsob vykonania, ktorý implikuje rozdielnu dĺžku spracovania a iné vyžadované zdroje. Tento problém je tiež modifikácia predošlej úlohy (*Job shop scheduling problem*).

1.3.1 Softvérové nástroje projektového plánovania

JIRA

JIRA je nástroj používaný na riadenie životného cyklu produktu a ponúka širokú paletu nástrojov, ktoré pomáhajú pri riadení vývoja produktu agilnými metodikami. Pomáha riadiť vývoj, plánovanie, stanovovať tok chýb a riadiť denné úlohy, sledovať postup tímu atď. V JIRA je tiež možné vytvárať reporty, sledovať, ako sa ktorý tím podieľa na zadaných úlohách, je možné sledovať predikcie vývoja či štatistiky minulých činností. JIRA podporuje agilné metodiky vývoja (Scrum, Kanban) a je veľmi flexibilná a konfigurovateľná. Podporuje celý proces vývoja v agilnom svete. Základné stavebné jednotky, ktoré je možné vytvárať a sledovať v JIRA sú:

Epic je obsahom najväčšia entita. Je to všeobecný prípad použitia aplikácie, ide o súbor funkcionalít (*User Stories*) a predstavuje relatívne veľký vývojový celok. Vyvinúť jeden *Epic* trvá spravidla niekoľko *sprintov*.

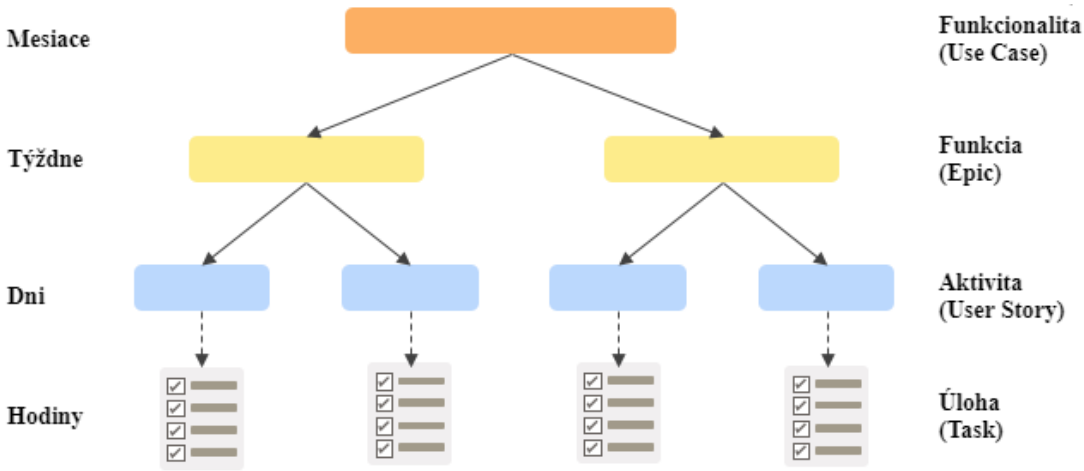
User Story reprezentuje používateľskú funkciu, je to systémová požiadavka zapísaná v netechnickom jazyku. Väčšinou sa *User Story* vyjadruje v štandardizovanom zápise "*Ako používateľ (kto konkrétne) chcem urobiť (ABC), aby som mohol urobiť (XZY)*". V časovom horizonte trvá vývoj *User Story* jeden *sprint*.

Task je úloha pre vývojára a predstavuje inžiniersku prácu, teda nemusí súvisieť s User Story. Môže sa jednať o technické podporné úlohy, ktoré nereprezentujú priamu hodnotu pre zákazníka, no je potrebné ich vykonať.

Sub-Task je pod-úloha pre vývojára a jej výsledkom sa priblíži celkový inkrement k dokončeniu vývoja *User Story*. Časovo sa vývoj *Sub-Taskov* meria v absolútnych jednotkých (najčastejšie v dňoch), a teda tieto pod-úlohy nemajú väčšinou priradené tzv. *Story points*.

Issue najčastejšie predstavuje chybu nájdenú v niektorej z vydaných verzií produktu. Pre *Issues* sa tiež nezadávajú spravidla časové odhady. [11]

Obrázok 3: Štruktúra základných elementov v JIRA [6]



[6] Zdroj: Epic, User Story, Task. [online]. Dostupné na internete: < https://www.visual-paradigm.com/scrum/theme-epic-user-story-task >

Veľmi podstatnou súčasťou JIRA sú nástenky resp. tabule (*dashboard, board*). Oba typy násteniek sú nastaviteľné podľa potrieb používateľa. Prvý typ, *dashboard*, slúži na zobrazovanie grafov, štatistických údajov a reportov. Druhý typ je závislý na metodike, ktorá je v projekte používaná. Existuje Scrum *board* a Kanban *board*, pričom každý obsahuje istý počet stĺpcov. Každý stĺpec predstavuje neusporiadaný zoznam úloh v určitom stave, napr. stav V zozname („*Sprint Backlog"*) (spomenutý v kapitole 1.2.3), V procese („*In Progress*") a Hotovo („*Done*"). Tieto stĺpce predstavujú stav, v akom sa nachádzajú úlohy, ktoré majú byť spracované. Keď sa stav úlohy zmení, úloha sa zobrazí v inom stĺpci. Takto je možné sledovať prehľadne priebeh práce na všetkých úlohách, ktoré boli zaradené do aktuálneho *sprintu*.

Scrumové tímy môžu metodiku používať aj bez softvérových nástrojov. Scrum tabuľu je možné vytvoriť pomocou nástenky, na ktorú sa pripínajú resp. lepia papierové štítky s popisom úloh. Výhodou softvérových nástrojov je však automatizácia a jednoduchšie spracovanie štatistík a ďalších potrebných dát o výkone tímu, či sledovanie predikcií ďalších iterácií.

OptaPlanner

OptaPlanner je nástroj, ktorý pomáha riešiť optimalizačné problémy a zároveň poskytuje používateľsky príjemné grafické rozhranie. Optimalizuje problémy plánovania a rozvrhovania, ako sú napríklad problém so smerovaním vozidiel (*Vehicle routing problem*), rozmiestnenie zamestnancov (*Employee rostering*), plánovanie údržby (*Maintenance scheduling*), priraďovanie úloh (*Task assigning*), problém rozvrhov (*School timetabling*), optimalizácia cloudu (*Clound optimization*), plánovanie schôdzok (*Conference scheduling*), *Job shop scheduling*, rozšírené úlohy o batohu a mnoho ďalších. [16] Každá organizácia čelí podobným výzvam, ako je prerozdelenie obmedzených zdrojov (zamestnanci, aktíva, čas a / alebo peniaze) na poskytovanie či výrobu svojich produktov alebo služieb. OptaPlanner pomáha nájsť čo najlepšie prípustné riešenie pre úlohy podobného typu.

OptaPlanner je rozšíriteľný nástroj, ktorý umožňuje Java programátorom efektívne riešiť NP-ťažké úlohy. Je tiež kompatibilný s inými jazykmi JVM (napríklad Kotlin a Scala). Je k nemu dostupná podrobná dokumentácia s ukážkovými príkladmi a umožňuje vytvárať vlastné modely.

OptaPlanner podporuje tri rodiny optimalizačných algoritmov: *Exhaustive Search*, konštrukčné heuristiky a metaheuristiky. V rodine *Exhaustive Search* algoritmov je metóda hrubou silou (*Brute force*) a metóda vetiev a hraníc. Tieto algoritmy dokážu nájsť optimálne riešenie, ale nie sú škálovateľné a nájdenie optimálneho riešenia môže trvať príliš dlho. Do rodiny konštrukčných heuristík patria metódy prvý vhodný (*First fit*), najslabší vhodný (*Weakest fit*), najsilnejší vhodný (*Strongest fit*), najlacnejšie vloženie (*Cheapest insertion*), *Regret insertion* a ich modifikácie. Medzi metaheuristiky, ktoré používa OptaPlanner, patria: simulované žíhanie (*simulated annealing*), tabu prehľadávanie (*tabu search*), horolezecký algoritmus (*hill climbing*), *late acceptance* a genetické algoritmy. [16]

Algoritmus, ktorý prehľadá celú množinu možností (napriek odrezaniu niektorých vetví ako v metóde vetiev a hraníc) môže problém reálneho sveta spracovávať milióny rokov. Cieľom je teda nájsť najlepšie možné riešenie v dostupnom čase. Na súťažiach zaoberajúcich sa plánovacími problémami (napr. *International Timetabling Competition*) sa ukázali variácie lokálneho prehľadávania (napr. Tabu prehľadávanie, simulované žíhanie, *Late acceptance*) ako metódy, ktoré poskytovali najlepšie výsledky pre plánovacie úlohy, ktoré mali byť spracované v dostupnom čase. [16]

V tejto diplomovej práci sa budeme zaoberať najmä dvoma typmi úloh, ktoré rieši OptaPlanner. Je to *Task assigning problem* a *Project job scheduling problem* spomenuté v predošlej kapitole.

Python, Python-MIP

Python ako programovací jazyk prináša so sebou mnoho prípadov použitia. Okrem webového vývoja a skriptovania je použiteľný aj pri spracovaní dát, strojovom učení, dátovej analýze či vizualizácii dát. Okrem toho sa Python často používa pri výučbe programovania a ponúka mnoho knižníc a balíčkov:

SciPy je *open source* knižnica pre vedecké a technické výpočty, obsahuje moduly na optimalizáciu, lineárnu algebru, integráciu, spracovanie signálu a obrazu, či *solver* na riešenie diferenciálnych rovník. (*Solver* je výpočtový softvérový nástroj alebo knižnica, ktorá rieši matematické úlohy.)

PuLP je *open-source* Python knižnica na modelovanie a riešenie problémov lineárneho programovania (LP). [17]

Python-MIP je nástroj na modelovanie a riešenie problémov zmiešaného celočíselného lineárneho programovaní (MILP). [8]

Google OR-Tools

OR-Tools resp. nástroje na operačný výskum je súbor *open source* nástrojov, ktoré pomáhajú riešiť úlohy lineárneho programovania, dopravné úlohy, grafové algoritmy a ďalšie problémy. [9] Pre úlohy lineárneho programovania používa Google OR Tools *open source slover* Glop. Pre ďalší typ úloh, ktorý hľadá prípustné riešenie na veľkej množine možných riešení (*constraint programming*), používa dva typy *solverov*, CP-SAT *solver* a CP *solver*. CP-SAT *solver* je technologicky vyspelejšie a robustnejšie riešenie, no CP *solver* rýchlejšie rieši malé príklady. Do tejto kategórie spadajú napríklad rozvrhovacie úlohy (*Employee rostering, Job shop problem*). Čo sa týka úloh celočísleného a zmiešaného celočísleného programovania, OR Tool využívajú MIP solver a CP-SAT *solver*. [10] V tejto diplomovej práci otestujeme riešenie *Job shop scheduling* problému pomocou Google OR Tools.

Roadmapa

Portfólio roadmapa je graficky znázornený plán smerovania relevantných komponentov, ktoré sa priamo dotýkajú strategických cieľov organizácie. [21] Je to živý dokument, ktorý by mal byť aktualizovaný pravidelne pri "re-optimalizácii" plánovania a pri hlavných zmenách v portfóliu a na jeho znázornenie sa podobá na Ganttov diagram.

Vďaka roadmape je možné sledovať zložité vzťahy medzi jednotlivými komponentami portfólia, pričom hlavným údajom je napĺňanie ich čiastkových cieľov v čase. Na časovej osi sú v roadmape zobrazené všetky dôležité míľniky v životnom cykle jednotlivých komponentov portfólia, od začiatku vývoja produktu, cez testovanie, schvaľovací proces, až po uvedenie produktu na trh. [21]

V agilnom prostredí môžu byť na roadmape znázornené podľa želanej granularity a želaných predpokladov dokončenia vývoja projektu znázornené *Epicy* (väčšie celky, ktorých vývoj trvá niekoľko *sprintov* a pozostávajú z *User Stories*),

alebo *User Stories* samotné. V našom prípade dávalo zmysel do Roadmapy zahrnúť aj jednotlivé *User Stories*, ktorých vývoj závisel aj na externých faktoroch (napr. vývoj *Backendu* alebo dostupnosť niektorých služieb od externých firiem).

Roadmapa pomáha pri plánovaní portfólia najmä vizuálne. Vďaka nej je možné intuitívne vnímať stav všetkých projektov a ich vývoj v čase - sledovať, v akej fáze by sa mal projekt momentálne nachádzať, a v akej fáze sa nachádza v skutočnosti. Nevýhodou roadmapy je, že neprináša detailnejší alebo technickejší pohľad, a teda podľa nej je náročné plánovať vývoj projektu z technického hľadiska.

Hlavnou výhodou roadmapy je, že je ľahko pochopiteľná pre manažment aj pre IT oddelenia. Sú na nej znázornené plánované *releasy* aplikácie a želania, kedy by mal ktorý projekt dokončiť svoj vývoj. Vďaka nej vieme relatívne ľahko odhadnúť, kedy budú ktoré tímy voľnejšie resp. viac vyťažené, kedy ktorý tím môžeme zapojiť do iného projektu atď. Nevýhodou roadmapy je, že vzniká ako súhrn želaní, očakávaní a predpokladov, teda neodzrkadľuje predpoklad vývoja na základe výpočtov či presnejších odhadov. Úplne sa spolieha na prvotné odhady dĺžky projektu prvotný odhad potrebných zdrojov.

1.4 Plánovanie vývoja v agilnom prostredí

Vývojárske tímy potrebujú časové odhady svojej práce, aby vedeli, ako dlho im potrvá doručiť daný produkt. Odhadovanie je náročné, pretože produktový *backlog* je často plný úloh už na mesiace dopredu, takže je možné urobiť len veľmi hrubý a neurčitý odhad. Tím si však postupne osvojí nejaké tempo, koľko jednotiek práce - hrubo odhadnutej - zvládne za *sprint*. Tomuto tempu sa hovorí "*velocity*".

Znamená to, že tím dokáže celkom presne odhadnúť, aké veľké porcie z *backlogu* treba zaradiť do *sprintu* tak, aby sa stihli dokončiť - aj napriek jednoduchým a hrubým odhadom. Aby tieto odhady náročnosti fungovali, tím musí pracovať s neustálou mierou neistoty. Tím potom pracuje na daných úlohách a zaznamenáva pri tom svoju činnosť. Vďaka záznamom o dokončených *sprintoch* tak vieme predpokladať, aká časť *backlogu* sa zmestí do ďalšieho *sprintu*.

Plánovanie a odhad úloh je nevyhnutné pre vývoj produktov iteratívnym spôsobom v súlade s požiadavkami špecifikovanými v Prioritizovanom produktovom *Backlogu*. Scrumový tím počas estimačných stretnutí odhaduje úsilie, ktoré budú vyžadovať jednotlivé úlohy v zozname úloh (*Task List*). Výstupom týchto stretnutí je zoznam úloh s odhadovanou námahou (*Effort Estimated Task List*). [2] V tomto zozname musí byť zahrnuté úsilie na testovanie a integráciu tak, aby výstupy *sprintu* mohli byť otestované a integrované do existujúceho produktu, ktorý bol výsledkom predošlých *sprintov*.

Stretnutia, na ktorých je odhadované potrebné úsilie, ktoré musí byť vynaložené na splnenie úloh, odhadujú aj kapacity tímov a ďalších potrebných zdrojov. Vďaka týmto predpovediam existuje spoločná dohoda, zdieľaná perspektíva, na *User Stories* a požiadavky a je možné predpovedať tzv. "*velocity*" *sprintu*.

Velocity určuje tempo vývoja produktu v tíme v relatívnych jednotkách. Rýchlosť vývoja jedného tímu za jeden *sprint* môže byť napríklad 40 "*Story points*", relatívnych jednotiek, ktoré tím priradzuje úlohám podľa ich náročnosti. V práci budeme používať pojem výkonnosť tímu s jednotkami *Story points* za iteráciu (SP/*i*).

Techniky odhadovania:

- dekompozícia,
- expertný posudok,
- analogické odhadovanie,
- parametrické odhadovanie.

1.4.1 Jednotky časových odhadov

Jednotky časového odhadu môžu byť absolútne (hodiny, dni, minúty) alebo relatívne (veľkosť tričiek, body). V Scrumovej metodike existuje pojem „ideálny čas", ktorý znamená počet hodín, ktoré developer venuje výlučne práci na vývoji výstupov, bez času ktorý potrebuje na iné aktivity alebo prácu mimo projektu. Ideálny čas je teda čistý čas potrebný na vývoj.

Story points vs. hodiny

Tradičné developerské tímy určujú odhady v časových formátoch: dni, týždne, mesiace. Agilné tímy však často používajú tzv. *Story points*. Mierka týchto bodov, tzv. *Story point rate* je relatívne úsilie práce vo formáte podobnom Fibonacciho číslam: 0, 0,5, 1, 2, 3, 5, 8, 12, 20, 40, 100. Na základe týchto bodov je možné vypočítať výkonnosť tímu. Výhoda používania *Story point* spočíva v tom, že členovia tímu sú odmeňovaní za riešenie úloh na základe náročnosti a je dodávaná hodnota a nie "odpracované hodiny".

Planning poker

Tím priradzuje náročnosť úlohám pomocou aktivity, ktorá sa volá *Planning poker*. Tím vezme položku z *backlogu*, stručne o nej diskutuje a následne každý člen urobí ohad náročnosti úlohy v relatívnych jednotkých (napr. *Story points*, veľkosť tričiek, atď). Potom všetci členovia ukážu karty so svojím odhadom. Ak sa všetci zhodli, táto úloha z *backlogu* dostane dané ohodnotenie. Ak nie, stručne sa diskutuje logický dôvod rodličných odhadov. Primárne je *Story points* možné použiť iba na úlohy typu *Story* alebo *Epic*, nie na *Bug*.

Nástenka v JIRA umožňuje používať nasledovné metriky:

- *Story points*,
- pôvodný časový odhad (minúty, hodiny, dni,..),
- počet úloh,
- veľkosť trička (XS, S, M, L, XL, XXL),
- iné, používateľom definované pole.

1.4.2 Velocity a Burndown chart

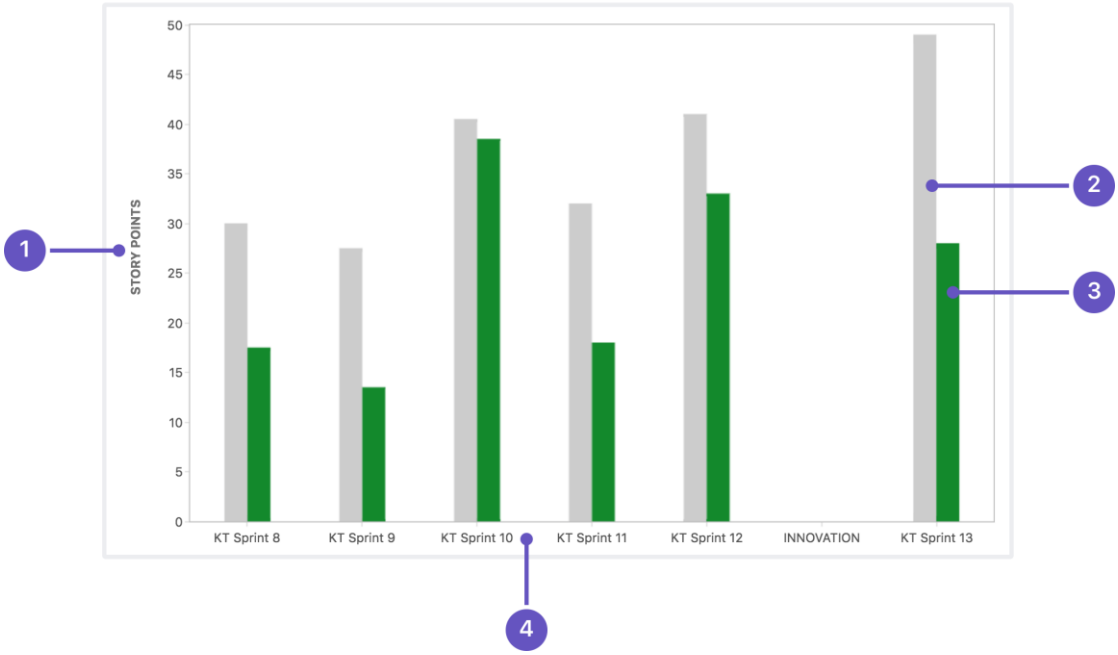
Velocity, ako sme spomínali v predošlej kapitole, je výkonnosť, resp. tempo tímu je založené na štatistike - pre každý *sprint* je tempo vlastne sumou odhadovanej náročnosti dokončených úloh. V JIRA sa nachádza graf s tempom tímu v "*Velocity Chart*". Do tejto štatistiky sa zarátavajú všetky úlohy, ktoré sú zaradené do *sprintu* - s

pôvodnou hodnotou. Ak sa teda zmení odhad náročnosti, táto zmena sa neprejaví vo výslednej štatistike.

Burndown chart je graf "spaľovania" a súvisí s tempom tímu. Odzrkadľuje prácu na daných úlohách v čase, resp. ako odbúda práca v danom *sprinte*. Ideálny stav je, aby na konci *sprintu* v *sprint backlogu* neostali žiadne úlohy, ktoré by neboli dokončené. Tento graf je možné nastaviť na relatívne jednotky (*Story points*) alebo čas. Časový graf však ukazuje iba čiastočný stav, a to počet hodín, ktoré boli odpracované a počet hodín, ktoré ostávajú v daný deň.

Na obrázku nižšie je znázornený graf tempa tímu. Zobrazuje množstvo práce, ku ktorej sa tím zaviazal že stihne daný *sprint* a porovnáva ju s reálnym výkonom tímu. Takto je znázornený každý *sprint*. Na x-ovej osi je množina ukončených *sprintov*, na y-ovej osi počet *Story points*.

Obrázok 4: Graf výkonnosti tímu počas jednotlivých *sprintov* [7]



Nasledujúce body vysvetľujú časti grafu znázornené na Obrázku 4.

1. Odhad v jednotkách: y-ová os na obrázku zobrazuje štatistiku vyjadrenú v *Story points*. Odhady môžu byť v iných jednotkách, absolútnych či relatívnych: počet hodín, počet úloh, či iných číselných hodnotách.

2. Záväzok (*commitment*): sivé stĺpce ukazujú množstvo práce, ktorú sa tím rozhodol odpracovať v daný *sprint*. Je to odhad všetkých úloh na začiatku *sprintu*. Akékoľvek zmeny po začiatku *sprintu* sa v tejto štatistike neprejavia.

3. Dokončené: zelené stĺpce predstavujú reálne množstvo dokončenej práce. Ak sa počas *sprintu* do *sprintu* pridajú ďalšie úlohy, zmena sa neprejaví v sivom stĺpci. Môže sa však stať, že počet dokončených úloh presahuje počet úloh zadaných v *sprinte*. Vtedy bude zelený stĺpec vyšší ako sivý.

4. *Sprinty*: x-ová os predstavuje posledných 7 dokončených *spintov*. Tieto dáta sa používajú na vypočítanie *velocity* tímu. [20]

---

[7] Zdroj: What is a product portfolio roadmap? [online]. Dostupné na internete: < https://aha.io/roadmapping/guide/product-roadmap/what-is-a-product-portfolio-roadmap >

1.4.3 Výpočet tempa

Výpočet tempa pre každý tím je užitočná pomôcka, ako odhadnúť množstvo práce, ktoré tím dokáže dokončiť v najbližšom *sprinte*. Tempo je vypočítané ako priemer všetkých dokončených odhadov počas posledných siedmich *sprintov*. Pre výpočet tempa pre tím, ktorého výsledky sú zobrazené na obrázku vyššie, používame nasledovný spôsob výpočtu aj v praktickej časti: 17.5 + 13.5 + 38.5 + 18 + 33 + 28) / 6 = 24,75; pričom prázdny *sprint* sa ignoruje. Znamená to, že nasledovný *sprint* môžeme odhadovať dodávku úloh v celkovom "objeme" 24,75 *Story points*. [20] Toto číslo sa stáva presnejším a spoľahlivejším až po čase, keď sa tím napríklad lepšie zladí resp. zoznámi sa s novým projektom.

V tejto kapitole sme ani zďaleka nevyčerpali všetky témy, o ktorých by sa dalo písať. Spomenuli sme tie, ktoré považujeme pre túto prácu najpodstatnejšie. Zamerali sme sa na popis optimalizačných problémov, ktoré sa týkajú plánovania. Popísali sme metódy projektového riadenia a detailnejšie sme sa zamerali na agilné metodiky a ich praktické použitie a terminológiu. V ďalšej kapitole predstavíme ciele a čiastkové ciele tejto diplomovej práce a ozrejmíme metódy skúmania.

2 Cieľ práce, metodika práce a metódy skúmania

Základným cieľom tejto diplomovej práce je preskúmať možnosti optimalizačných nástrojov, ktoré sa dajú využiť pri plánovaní projektového portfólia. Keďže ide o NP-ťažkú úlohu, zatiaľ neexistuje nástroj, ktorý by dokázal vývoj projektu či projektového portfólia naplánovať optimálne vzhľadom na rozsah úlohy, veľké množstvo zdrojov, premenných, podmienok a kombinácií riešení daného problému v primeranom čase.

Našim cieľom je preto otestovať vhodné nástroje na reálnom príklade. Čiastkovým cieľom je porovnať ich použiteľnosť a výsledky, ktoré tieto nástroje dokážu priniesť. Snažíme sa hľadať nástroj, ktorý poskytne najvyššiu pridanú hodnotu množstvom a obsahom výsledných informácií, ktoré potom môžu slúžiť na informované manažérske rozhodnutia, no zameriavame sa aj na jednoduchosť implementácie a použitia daného nástroja.

Porovnávame rôzne typy rozvrhovacích úloh s príbuzným cieľom, a to rozvrhnutie úloh z Produktového *backlogu* scrumovým tímom tak, aby výsledný čas spracovania všetkých úloh bol minimálny. Vybraný vhodný nástroj následne aplikujeme pri riešení plánovania projektového portfólia v agilnom veľkoformátovom prostredí so šiestimi tímami a úlohami, ktoré je potrebné naplánovať a niekoľko mesiacov dopredu.

Čiastkovým cieľom je nájsť také riešenia, ktoré dostatočne realisticky simulujú rozdielne vlastnosti a schopnosti tímov a zohľadňujú ich aj pri spracovaní úlohy. Ďalším čiastkovým cieľom je porovnanie zvolených nástrojov a ich možností. Hlavným cieľom je aplikácia vybraného nástroja na zadaný problém a nájdenie optimálneho resp. najlepšieho možného riešenia, ako naplánovať vývoj zadaných projektov s dostupnými zdrojmi.

Ako vstupné dáta pre túto úlohu slúžili dáta exportované z JIRA projektu, ktorý funguje ako nástroj na riadenie celého LeSS tímu. Pre prvotné otestovanie nástrojov sme použili obmedzenú množinu dát a simulovali sme rozdelenie 10 úloh (spolu 320 *Story points*) medzi 3 tímy. Pre vybraný nástroj a finálne riešenie boli použité všetky úlohy, ktoré boli v JIRA ohodnotené.

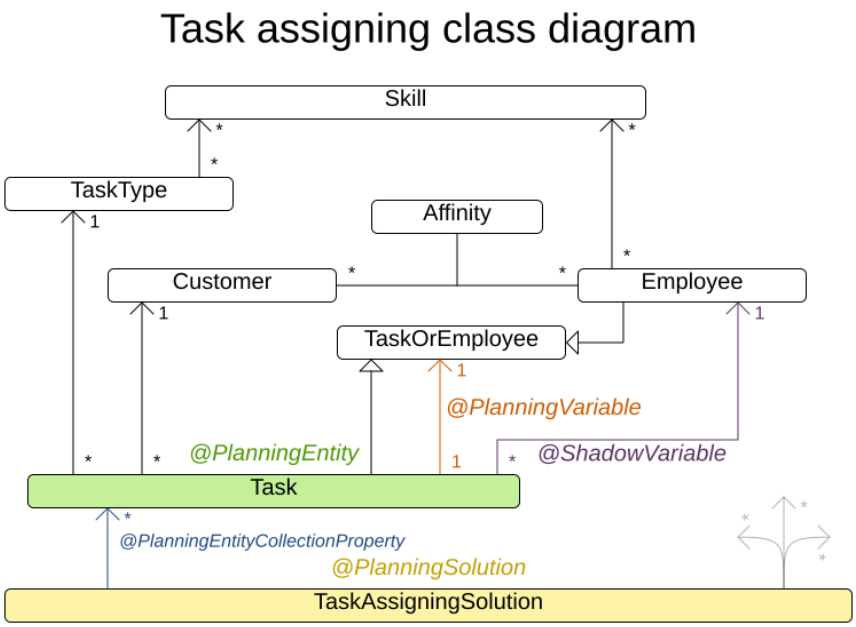
Takýchto úloh sa tam nachádzalo 135, boli ohodnotené na 1932,125 *Story points*, pričom jeden tím priemerne za jednu iteráciu dokáže vypracovať úlohy s objemom 22,5 *Story points*.

Z nástrojov a úloh, ktoré spomíname v kapitole 1.3, sme sa rozhodli otestovať nasledovné možnosti:

2.1 OptaPlanner a Task assigning problem

Ako prvú úlohu riešime priraďovací problém. Popis tohto problému je uvedený v kapitole 1.3.1 Najčastejšie druhy problémov a úloh v procese plánovania. Doménový model pre túto úlohu je v OptaPlanneri zobrazený na Obrázku 5. Existuje trieda *Employee*, zamestnanec, ktorá predstavuje vykonávateľa danej úlohy. Ďalšia trieda, *Customer*, je zákazník, ktorý danú úlohu požaduje resp. si ho môžeme predstaviť ako zadávateľa úlohy. *Task* je úloha, *Skill* reprezentuje zručnosť daného zamestnanca, ktorá je potrebná na dokončenie úlohy, napríklad zručnosti s MS Word, ovládanie cudzieho jazyka a podobne. *Affinity* znamená príbuznosť zamestnanca k danému zákazníkovi. Ak má zamestnanec vyššie skóre príbuznosti, znamená to, že bude úlohu pre daného zákazníka vykonávať kratší čas, ako zamestnanec, ktorý má skóre afinity nižšie. *Task Type* popisuje typ úlohy, napríklad spracovanie daní alebo interview.

Obrázok 5: Doménový model pre *Task assigning problem* [8]



Do úvahy sa berú tzv. *hard* a *soft constraints*, slabo a silno obmedzujúce podmienky, ktoré môžu byť rozdelené na viacero úrovní. [14] Silno obmedzujúca alebo striktná podmienka musí byť splnená, zatiaľ čo slabo obmedzujúca podmienka môže byť porušená, ale jej splnenie pozitívne prispieva k hodnoteniu daného riešenia. Tieto podmienky, na základe ich charakteru, môžu alebo nemôžu byť porušené pri hľadaní optimálneho resp. najlepšieho možného riešenia. Príklad podmienok:

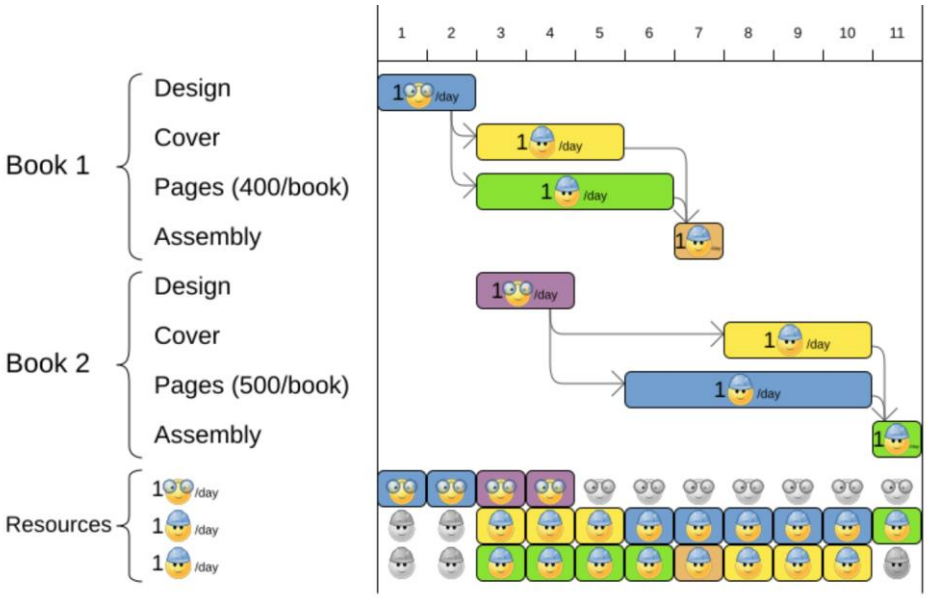
- **Silno obmedzujúca podmienka:** Každá úloha vyžaduje na splnenie jeden alebo viacero zručností. Na to, aby mohol zamestnanec vykonať danú úlohu, musí ovládať všetky tieto zručnosti.
- **Slabo obmedzujúca podmienka:** Úlohy s najvyššou prioritou by mali byť dokončené skôr, ako úlohy s nižšou prioritou.

[8] Task Assigning Problem. [online]. s.64. Dostupné na internete: < https://docs.optaplanner.org/7.37.0.Final/optaplanner-docs/html_single/index.html#taskAssigning >

2.2 OptaPlanner a Project job scheduling problem

Cieľom *Project job scheduling* problému je minimalizovať zdržania na každom projekte a zároveň minimalizovať celkovú dĺžku všetkých projektov. Tento model umožňuje plánovanie so zdieľanými, ako aj s lokálnymi zdrojmi. Umožňuje plánovanie viacerých projektov, ide o formu flexibilného *Job shop scheduling* problému. Zdroje môžu byť obnoviteľné a neobnoviteľné. Grafické znázornenie problému je na Obrázku 6.

Obrázok 6: Grafické znázornenie problému [9]



2.3 Knižnica MIP a Job shop scheduling problem

MIP Knižnica je Python knižnica, ktorá poskytuje programátorské riešenie NP-ťažkého problému *Job shop scheduling*. Motivácia je spracovanie úloh na viacerých strojoch, za podmienky, že každá úloha musí byť každým strojom spracovaná práve jedenkrát a toto spracovanie trvá určitý čas.

Okrem toho, pre každú úlohu je určené poradie strojov, na ktorých má byť vykonávaná. Matematický zápis úlohy je zobrazený nižšie.

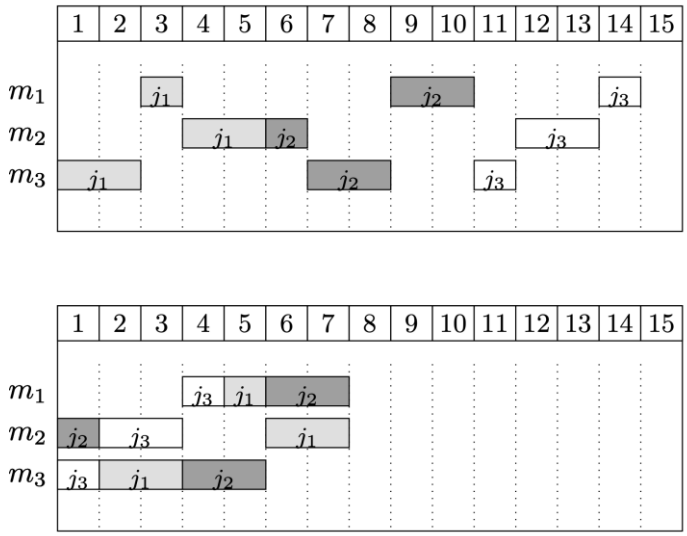
[9] Zdroj: Project Job Scheduling. [online]. Dostupné na internete: < https://docs.jboss.org/drools/release/6.0.0.Final/optaplanner-docs/html_single/index.html#projectJobScheduling >

Obrázok 7: Matematický model Job shop scheduling problému [10]

$$
\begin{aligned}
x_{o^j_r j} &\geq x_{o^j_{r-1} j} + p_{o^j_{r-1} j} \;\; \forall r \in \{2, .., m\}, j \in \mathcal{J} \\
x_{ij} &\geq x_{ik} + p_{ik} - M \cdot y_{ijk} \;\; \forall j, k \in \mathcal{J}, j \neq k, i \in \mathcal{M} \\
x_{ik} &\geq x_{ij} + p_{ij} - M \cdot (1 - y_{ijk}) \;\; \forall j, k \in \mathcal{J}, j \neq k, i \in \mathcal{M} \\
C &\geq x_{o^j_m j} + p_{o^j_m j} \;\; \forall j \in \mathcal{J} \\
x_{ij} &\geq 0 \;\; \forall i \in \mathcal{J}, i \in \mathcal{M} \\
y_{ijk} &\in \{0, 1\} \;\; \forall j, k \in \mathcal{J}, i \in \mathcal{M} \\
C &\geq 0
\end{aligned}
$$

Možné riešenia jednoduchej úlohy pre rozdelenie troch úloh a troch strojoch je znázornené na Obrázku 8.

Obrázok 8: Naivné a optimálne riešenie úlohy [11]



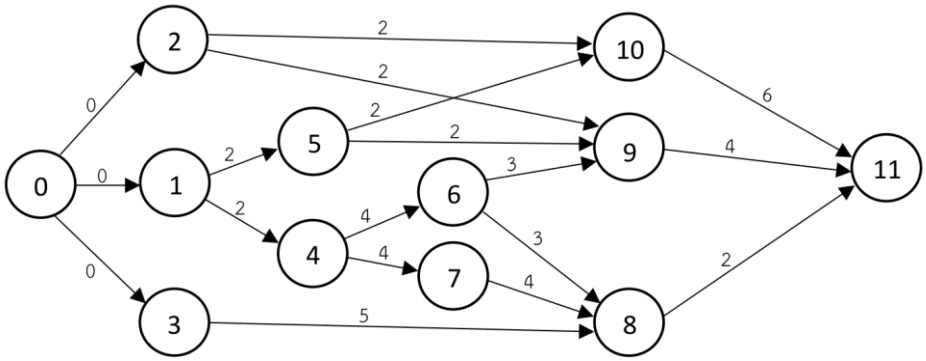
[10] Zdroj: SANTOS, Haroldo G. – TOFFOLO, Túlio A.M. Mixed Integer Linear Programming with Python. 2019. s 20. [online]. Dostupné na internete: < https://buildmedia.readthedocs.org/media/pdf/python-mip/latest/python-mip.pdf >

[11] Zdroj: SANTOS, Haroldo G. – TOFFOLO, Túlio A.M. Mixed Integer Linear Programming with Python. 2019. s 19. [online]. Dostupné na internete: < https://buildmedia.readthedocs.org/media/pdf/python-mip/latest/python-mip.pdf >

2.4 Knižnica MIP a úloha plánovania projektu s obmedzením zdrojov

Úloha plánovania projektu s obmedzením zdrojov alebo *Resource constrained project scheduling problem*, je kombinatorický optimalizačný problém a pozostáva z hľadania prípustných riešení pre množinu *n* úloh vzhľadom na zdroje a podmienky poradia. Každá úloha môže začať až vtedy, keď sú všetky predchádzajúce úlohy dokončené. Preto je možné úlohu zakresliť ako graf. Nultý a jedenásty uzol sú prázdne uzly, ktoré predstavujú počiatok a koniec plánovania. [8] Ide o hranovo ohodnotený graf, pričom hodnota hrany predstavuje dĺžku vykonávania danej úlohy, spotreba zdrojov sa uvádza v tabuľke ako hodnota každého uzlu.

Obrázok 9: Grafické znázornenie problému plánovania projektu s obmedzením zdrojov[12]



2.5 Google OR Tools a Job shop scheduling problem

Ide o rovnaký problém, ako sme popísali pri popise MIP knižnice v časti 2.3, z tohto dôvodu už samotný problém nebudeme popisovať. Popíšeme však stručne charakteristiku nástroja Google OR Tools. V tomto prípade nám Google OR Tools ponúka možnosť vybrať si jazyk, v ktorom chceme daný problém implementovať. Je možné použiť Python, Javu a C#.

[12] Zdroj: SANTOS, Haroldo G. – TOFFOLO, Túlio A.M. Mixed Integer Linear Programming with Python. 2019. s 17. [online]. Dostupné na internete: < https://buildmedia.readthedocs.org/media/pdf/python-mip/latest/python-mip.pdf >

OR Tools používa dva *solvery*, CP-SAT a originálny CP *solver*. CP-SAT *solver* je technologicky nadradenejší originálnemu CP *solveru* a mal by byť preferovaný v takmer všetkých situáciách. Výnimkou sú malé problémy, pre ktoré môže byť riešenie nájdené rýchlo. V takomto prípade CP *solver* prekoná CP-SAT *solver*. [10]

Týmto sme vyčerpali tému predstavovania použitých metód a v nasledujúcej kapitole sa zameriame na praktickú aplikáciu spomínaných problémov pomocou týchto softvérových nástrojov.

3 Výsledky práce a diskusia

Túto kapitolu sme rozdelili na dve hlavné časti. V prvej sa venujeme popisu výsledkov čiastkových cieľov, čiže výberu vhodného nástroja na použitie plánovania projektového portfólia. Jednotlivé nástroje testujeme na obmedzenej množine vstupných dát, ktoré pochádzajú z reálneho príkladu z praxe. V tejto časti uvádzame tiež iné možnosti použitia daného riešenia v projektovom plánovaní a porovnanie výhod a nevýhod každého riešenia, čo bol tiež jedným z čiastkových cieľov tejto práce.

V druhej časti sa zameriavame na vybraný nástroj, na jeho implementáciu na príklad plánovania portfólia a popis výsledkov aplikácie. Popisujeme hlavný cieľ a prínos tento diplomovej práce.

3.1 OptaPlanner, JAVA implementácia

V tejto podkapitole popisujeme aplikáciu dvoch problémov riešených v nástroji OptaPlanner, a to problém priraďovania úloh a problém rozvrhovania projektových úloh. Oba tieto príklady sme považovali za vhodné na modelovanie zadaného problému, pretože už samotný koncept obidvoch problémov je zrozumiteľný a sám sa ponúka ako možné riešenie pre plánovanie projektového portfólia. V ďalších dvoch podkapitolách vysvetľujeme, ako boli dané modely aplikované, a aké ich výhody a nevýhody vnímame.

3.1.1 Task assigning problem

Pri implementácii tejto úlohy sme trochu pozmenili doménový model, resp. význam jednotlivých tried modelu, ktorý je spomínaný v kapitole 2.1 a prispôsobili sme ho našim potrebám. Zmeny v doménovom modeli sú vypísané v Tabuľke 1.

Tabuľka 1: Doménový model a jeho zmeny pre *Task assigning problem*

Pôvodný model	**Nový model**
Employee	Tím
Customer	Projekt Pôvodne boli tímy pridelené na projekty.
Task	Task Predstavuje úlohu v Backlogu. Má prioritu.
Skill Schopnosť Employee	Zručnosť Požadované schopnosti tímu.
Affinity Príbuznosť zamestnanca k zákazníkovi.	Príbuznosť k projektu Niektoré tímy si vypomáhali a vidia aj do iných častí aplikácie.

Zmeny modelu neboli zásadne veľké, primárne sme pozmenili sémantický význam úlohy, ale zachovali sme obsahový model. V pôvodnom modeli boli úlohy priraďované jednotlivým zamestnancom a každý z nich mal určitú sadu zručností. V našom prípade dávalo väčší zmysel priraďovať úlohy konkrétnym tímom, ktorých zručnosti sme definovali ako súhrn zručností všetkých členov tímu.

Ďalšou zmenou v doménovom modeli bola potreba identifikácia toho, kto je vlastne zákazník, resp. k akej entite má prislúchať vzťah každého tímu vzhľadom na ich príbuznosť. Ako odpoveď sa naskytovali dve možnosti. Po prvé, mohli sme na miesto zákazníka prehlásiť jednotlivé komponenty aplikácie, ktoré vyžadujú určité znalosti pri obsluhe, takže modelovanie afinity by bolo jednoduché, no počet zákazníkov by bol omnoho väčší. Ako druhá možnosť sa naskytoval práve projekt, čo do veľkej miery reflektovalo realitu.

Pôvodne boli tímy vytvárané v čase, keď vznikla požiadavka na nový projekt, preto aj svoj čas členovia tímu primárne venovali vývoju daného projektu, čím vznikala afinita medzi tímom a projektom. Po čase mohol tím dostať na starosti údržbu iného projektu alebo prejsť na vývoj nového projektu. Dôležitý bod, ktorý je nutné spomenúť, je aj to, že aj v prípade, že nejaký tím zanikol, členovia scrumového tímu ostali ako súčasť portfólia a buď sa vytvoril nový tím, alebo boli títo členovia prerozdelení do iných tímov. To znamená, že každý tím mohol mať skúsenosti z viacerých projektov a know-how nadobudnuté na projektoch ostávalo medzi vývojármi.

Vďaka atribútu príbuznosti je tak možné priradiť úlohy aj iným, v danej oblasti menej skúseným tímom. Na základe zručností spracovávateľa môže byť pôvodná dĺžka spracovania úlohy (v modeli označená ako *Base Duration*) predĺžená o adekvátny čas. Táto charakteristika modelu je veľmi realistickým odzrkadlením skutočnosti. Každý tím by teoreticky dokázal vyvinúť akúkoľvek funkcionalitu. V prípade, že na podobnej úlohe už tím pracoval, bude to trvať kratší čas, ako v prípade nového tímu. Logickou úvahou teda vyplýva, že snaha *solvera* je priradiť úlohy tímom, ktoré majú všetky potrebné zručnosti a zároveň najvyššiu mieru príbuznosti so zadávateľom úlohy.

Ďalšou výhodou je čitateľná a zrozumiteľná štruktúra súboru pre vstupné dáta. Jedná sa o formát XML, ktorý dokáže hierarchicky popísať vzťahy medzi všetkými entitami (tím, projekt, zručnosť, atď.) vstupujúce do úlohy.

Implementácia tohto riešenia vyžaduje schopnosť pracovať s jazykom JAVA v tom prípade, že potrebujete upraviť alebo rozšíriť základnú úlohu o väčšie zmeny. V prípade konfiguračných nastavení (typ *solvera*, maximálny čas spracovania) je potrebné tieto zmeny zapísať len v konfiguračnom XML súbore. OptaPlanner ponúka obšírne popísanú dokumentáciu, podľa ktorej by to mal zvládnuť aj človek bez programátorských zručností.

Jednou z nevýhod je potreba manuálne pripravovať vstupné dáta, resp. naprogramovať vlastné riešenie, ktoré by konvertovalo exportovaný súbor z JIRA do požadovaného formátu XML v OptaPlanneri. JIRA poskytuje relatívne veľký priestor na konfiguráciu želanej množiny dát a formátu súboru. Bohužiaľ v OptaPlanneri neexistuje nástroj, ktorý by dokázal importovať v inom formáte a následne ich konvertovať.

OptaPlanner dokáže využívať kombináciu rôznych heuristík ako sme spomínali v kapitole 1.3.2. V tomto prípade však neboli použité žiadne heuristiky, ktoré by simulovali učenie tímov. Ak tím pracuje na úlohách, ku ktorým nemá najvyššiu mieru príbuznosti, časom by mala táto miera príbuznosti vzrásť. Každý tím by tak v čase mal priestor získavať nové skúsenosti a tieto by sa premietali aj do ďalších výsledkov v reálnom čase spracovania úlohy. V našom prípade bol však výpočet dĺžky spracovania úlohy pevne daný vstupnými dátami, ktoré sa počas behu programu nemenili.

3.1.2 Project job scheduling

Ako uvádzame v kapitole 2.1, cieľom úlohy *Project job scheduling* je minimalizovať meškanie každého projektu a zároveň celkovú dĺžku prác na všetkých projektoch. Problém berie do úvahy poradie úloh, pričom nasledovná úloha sa nezačne vykonávať, pokiaľ predošlá nie je dokončená. Úloha uvažuje aj kapacitné obmedzenia a podmienky zdrojov, ktoré sú lokálne (zdieľané medzi úlohami na rovnakom projekte) alebo globálne (zdieľané medzi všetkými úlohami), a zdroje môžu byť obnoviteľné (určené na deň) alebo neobnoviteľné. Snaha je minimalizovať trvanie každého projektu, resp. celého portfólia projektov, tzv. *multi-project schedule*, resp. viacprojektový rozvrh.

Tento model asi najlepšie zo všetkých vybraných úloh modeluje skutočnú povahu projektového portfólia a jeho riadenia, ako aj manažérske zámery minimalizovať časové rozpätie projektov. Žiaden z testovaných modelov totiž explicitne nepopisuje potrebu zdieľania kapacít medzi rôznymi projektami, ktoré majú rozdielne kapacitné potreby a termíny. V portfólio manažmente je navyše kritická úloha prerozdelenia dostupných kapacít, v našom prípade sa jedná o scrumové tímy a ich členov, na jednotlivé úlohy.

V realite je možné za lokálne zdroje považovať tých členov tímu, ktorí majú buď veľmi špecifické znalosti, ktoré nie sú vyžadované na žiadnom inom projekte, alebo nie sú ešte dostatočne skúsení a tak nevzniká opodstatnenie vypomáhať v iných tímoch v prípade potreby, čo je bežná prax. Za zdieľané zdroje by sme mohli považovať napríklad potrebu *Backendových* vývojárov, ktorí nemajú uplatnenie v každom tíme počas celého *sprintu*, resp. testerov, ktorí sú síce súčasťou scrumového tímu, no v skutočnosti musia testovať a mať prehľad o celej aplikácii. V prípade, že sa niektorá verzia teda zameriava výlučne na novú funkcionalitu, sú potrebné všetky kapacity testerov zo všetkých tímov. V prípadovej štúdii však nepotrebujeme riešiť zdieľané kapacity testerov naprieč tímami, pretože sa zameriavame iba na vývoj nových funkcionalít a nie na opravu *bugov*.

Ďalšou výhodou tohto modelu je samozrejme grafické rozhranie, ktoré zobrazuje priebeh riešenia v reálnom čase, ako aj výsledné riešenie. Táto vlastnosť sa však dotýka každého problému, ktorý je možné riešiť v nástroji OptaPlanner.

Oproti Task assigning problému sa v tomto modeli bohužiaľ upúšťa od zavedenia premennej, ktorá určovala príbuznosť kapacít ku konkrétnej činnosti projektu. V realite by tak bolo náročné modelovať prípady napríklad rozdielnej seniority členov tímu, kedy by boli požadované napríklad zručnosti analytika, ale nemohli by sme jednoznačne rozlíšiť, či ide o potrebu juniora, mediora alebo seniora.

Z pohľadu Scrumovej metodiky je tento prístup správny. Členovia tímu by sa mali podieľať na vývoji aplikácie rovnako, ich schopnosti medzi tímami by sa nemali značne líšiť, teda nemali by vznikať „špecializované tímy“. V teórii by mal mať teda každý tím schopnosť pracovať na ktoromkoľvek *tasku*, ktorý je v *Backlogu*. V našom prípade to tak však nie je, pretože každý tím bol pôvodne vyčlenený na prácu na inom projekte, a teda logicky je lepší v programovaní niektorých typov úloh.

Ďalšou, a to najväčšou nevýhodou, je formát vstupných dát. Dáta sú rozdelené v troch formátoch súborov (xml, txt, a mm) pričom nie sú zdokumentované a ľudsky čitateľné, z čoho vyplýva veľmi náročná možnosť editovať vstupné dáta. Takto spracovať dáta pri každom plánovaní by vyžadovalo naprogramovanie si vlastného nástroja na migráciu dát, ich prepis a vygenerovanie príslušných súborov, čo nie je pre naše potreby prijateľná cesta, preto sme v hlbšom skúmaní tohto problému nepokračovali.

Celkovo by sme výsledok testovania tohto problému zhrnuli nasledovne: *Project job scheduling* reálne popisuje problematiku plánovania projektového portfólia. Jeho nevýhoda je v tom, že abstrahuje od schopností tímov resp. príbuznosti tímov k projektu, čo ovplyvňuje dĺžku vývoja jednotlivých úloh. Takto by poskytol najlepšie možné, resp. optimálne riešenie, ale nereflektovalo by vývoj v realite. Ešte väčšia nevýhoda je komplikovanosť štruktúry vstupných dát, čo je jedným z hlavných dôvodov, prečo sme tento nástroj nezvolili.

3.2 MIP Knižnica

3.2.1 Job shop scheduling problem

Keďže ide iba o knižnicu písanú v Pythone, ktorá ponúka pomoc pri programovom riešení optimalizačných úloh, od tohto riešenia nemôžeme očakávať

žiadnu podporu grafického riešenia bez toho, aby sme ho sami implementovali. Preto je tiež potrebné celé riešenie naprogramovať. Z tohto vychádza potreba pracovať na úrovni programátorského kódu, a teda aj vstupné dáta je potrebné zadávať v štruktúre zrozumiteľnej pre model tohto problému.

Model vstupných dát je potrebné rozdeliť na dve časti. Prvá časť sú dáta, ktoré vyjadrujú čas potrebný na spracovanie jednotlivých úloh na daných strojoch. Pri vývoji softvéru je častokrát potrebné, aby úloha prešla postupne viacerými fázami a za každú fázu môže byť zodpovedný iný riešiteľ. Napríklad úloha prejde v JIRA viacerými fázami, ako spomíname v kapitole 1.3.2 a tieto fázy môže predstavovať fáza analýzy, vývoja a testovania, pričom každú fázu môže obhospodarovať iná skupina členov tímu. V našom prípade uvažujeme nad tímami, ktoré majú v sebe členov na rôznych pozíciách (tester, vývojár, analytik), preto je každý tím sebestačný a nie je potrebné úlohu presúvať medzi tímami. Ak by sme takéto riešenie chceli vytvoriť, úlohy by sme priraďovali tímom testerov, tímom vývojárov alebo tímom analytikov, ktoré v LeSS metodike fungujú pod názvom *Community of Practice*. Čas spracovania pre daný tím je teda určený pozíciou v príslušnom poli, ostatné časy spracovania sme nechali nulové.

Obrázok 10: Formát vstupných údajov v knižnici MIP [13]

```
times = [[0, 0, 13],
         [0, 13, 0],
         [100, 0,0],
         [20, 0, 0],
         [0, 0,100],
         [40, 0, 0],
         [8, 0,  0],
         [0, 0,  1],
         [0, 20, 0],
         [5, 0,  0]]
```

Druhá časť dát je vkladaná do podobnej štruktúry. Ide o usporiadanie, v akom danú úlohu spracujú jednotlivé stroje. Každý riadok reprezentuje jedna úloha ako pole,

[13] Zdroj: Autor

prvky poľa predstavujú indexy strojov. Na základe toho vieme vždy priradiť, ktorá úloha má byť priradená ktorému stroju a na aký dlhý čas.

Keďže ide o knižnicu, takže na prácu s ňou je potrebná znalosť programovania. Táto skutočnosť môže pôsobiť ako nevýhoda v prostredí manažérskeho plánovania projektov. V skutočnosti však vidíme priestor tieto knižnice použiť na vytvorenie samostatného nástroja, ktorý bude slúžiť presne na zadané účely.

Jednou z praktických nevýhod pri hľadaní jednoduchého a ľahko použiteľného riešenia je však potreba konvertovať všetky dáta na pole polí s pevne stanovenou dĺžkou pre všetky vnorené polia. To znamená, že sa očakáva, že každá úloha musí byť spracovaná každým strojom. V skutočnosti však vzniká potreba predávať úlohy flexibilnému počtu strojov, čo v tomto modeli nie je možné. Úlohu sa nám podarilo nasimulovať aj napriek tomu, vznikla však tzv. riedka matica (*sparse matrix*), kedy dátová štruktúra obsahovala veľké množstvo nulových hodnôt. Z toho dôvodu aj spracovanie úlohy trvalo dlhšie, pretože *solver* musel plánovať aj úlohy s nulovou dĺžkou.

Táto úloha je teda nevhodná pre modely, kedy vieme, že jedna úloha má byť spracovaná iba jedným strojom resp. flexibilným počtom strojov. Ide o prípady, kedy očakávame, že jeden *Epic* má byť spracovaný výlučne jedným tímom. Dátový model musí byť doplnený o nulové časy spracovania úlohy pre všetky ostatné stroje, ale algoritmus berie tieto premenné do úvahy pri výpočte, čo predlžuje čas spracovania úlohy a ovplyvňuje výsledky.

Model považujeme za vhodný v prípade, keď vieme, že všetky úlohy musia byť riešené viacerými tímami. Model by bol vhodný pri menšej granularite, teda ak by sme do úvahy nebrali celé tímy ale iba časti tímu a jeden stroj by bola skupina členov tímu, ktorí sa zaoberajú jednou vývojovou fázou softvéru (analytici, developeri, testeri). Vtedy by každá úloha musela prejsť najskôr fázou analýzy, potom fázou implementácie a nakoniec fázou testovania.

Knižnica využíva primárne MIP *solver* CBC, nie komerčný Gurobi *solver*, hoci je kompatibilná aj s ním. Každý pokus pritom vypisuje na obrazovku, takže dĺžka spracovania úlohy sa predlžuje. Oproti knižnici Google OR Tools je tak niekoľkonásobne pomalší, a to aj pri špecifickom type úlohy.

MIP knižnica pre testovací príklad našla optimálne riešenie, celkový čas spracovania úloh je 173 *Story points* a celkový čas bol 13.27 sekúnd. Výsledok bol vypísaný v konzole nasledovným spôsobom zobrazeným na Obrázku 11.

Obrázok 11: Výsledné riešenie problému v knižnici MIP [14]

```
Výsledok: nájdené optimálne riešenie
Hodnota účelovej funkcie:       173.00000000
Vymenované uzly:                1430
Počet iterácií:                 91826
Celkový čas (CPU v sekundách)   13.27
Čas ukončenia:             173.0

úloha 1 začne na stroji 1 v čase 0
úloha 1 začne na stroji 2 v čase 40
úloha 1 začne na stroji 3 v čase 60
...
úloha 10 začne na stroji 1 v čase 0
úloha 10 začne na stroji 2 v čase 60
úloha 10 začne na stroji 3 v čase 73
```

3.2.2 Úloha plánovania projektu s obmedzením zdrojov

Tento problém je definovaný pomocou grafovej štruktúry. Každý uzol predstavuje úlohu a každá hrana predstavuje prechod z jednej úlohy do druhej. Každý uzol má svojho predchodcu a svojho nasledovníka. Existuje jeden počiatočný bod úlohy a jeden koncový bod úlohy. Pre tieto existuje tzv. prázdny uzol. V našom prípade, keď pracujeme s 10 úlohami, je tento uzol v poradí nultý a jedenásty.

Keďže graf musí byť hranovo ohodnotený, cena hrany medzi jedným a druhým uzlom predstavuje časovú náročnosť spracovania úlohy. Navyše, každá úloha má potrebu využiť isté množstvo zdrojov. Zdroje sú považované za obnoviteľné a pre každý uzol je potrebné iné množstvo zdrojov. Tieto zdroje môžu byť rôzneho typu.

[14] Zdroj: Autor

Pre modelovanie plánovania vývoju projektu sme preto určili tri typy zdrojov. Ide o skupinu analytikov, skupinu Android vývojárov a skupinu iOS vývojárov.

Takto zadaný model relatívne dobre vystihuje podstatu plánovania a pracuje najmä so zdrojmi, v čom je jeho obrovská výhoda. V prípade, že sa snažíme minimalizovať náklady na tím, čo je v praxi bežná situácia, je možné tento model využiť na optimalizáciu zdrojov. Model môže ostať nezmenený a stačí úlohu otestovať s rôznymi vstupnými dátami pre zdroje. V takom prípade vieme porovnať, ako dlho bude trvať dokončenie projektu pre rôzny počet špecialistov a dokážeme si vybrať riešenie, ktoré minimalizuje náklady no jeho dokončenie je stále v požadovanom časovom rozpätí.

Ďalšia výhoda tohto modelu je, že relatívne flexibilne umožňuje zadávať počty skupín, resp. tímov. Na základe zmien týchto údajov tak vieme model využiť aj na predpoklad efektivity práce a ceny vývoja za určité časové obdobie.

Jeden z atribútov, ktoré v tejto úlohe nerešpektujeme, je však agilný vývoj. V agilnom vývoji tiež záleží na určitých závislostiach vo vývoji niektorých komponentov softvéru, ale poradie úloh nie je striktne definované a je možné pracovať na ktorejkoľvek úlohe v *Scrum backlogu*. V tejto úlohe je každá úloha reprezentovaná ako uzol v grafe, ktorý presne určuje náväznosti. Do určitej miery je aj toto prípustné a na strane používateľa je možné usporiadať úlohy do grafu podľa priority tak, aby boli najdôležitejšie úlohy spracované ako prvé. Úlohu plánovania projektu s obmedzením zdrojov však považujeme za vhodnú pre projekty vedené ako vodopádový model.

Toto riešenie ponúka priestor pre finančné odhady a plánovanie, pokiaľ existuje presne spracovaný plán náväznosti jednotlivých úloh. V opačnom prípade by si tvorba takéhoto modelu na príklade z praxe vyžadovala časovo extrémne náročnú prácu analytika, ktorý musí poznať každý detail všetkých úloh, aby ich dokázal správne zoradiť a odhadnúť ich časovú náročnosť a požiadavky na zdroje.

Toto riešenie preto nepovažujeme za škálovateľné a vhodné pre veľké projekty. Pre plánovanie portfólia projektov by sme tak nemohli spracovať na úrovni jednotlivých úloh ale s väčšou granularitou, napríklad na úrovni *Epicov* resp. celých projektov. Okrem toho, vývojové tímy strávia určitú časť *bugfixingom*, a teda pri optimalizácii nákladov by sme museli vziať do úvahy okrem samotného riešenia aj určitú rezervu.

Čo sa týka detailného plánovania jednotlivých zdrojov, riešenie tiež nereflektuje úplne skutočné správanie Scrumových tímov.

V prípade, že sa zdroj uvoľní z jednej úlohy skôr, môže začať pracovať na ďalšej úlohe. V tomto prípade sa vždy čaká na uvoľnenie všetkých zdrojov potrebných na splnenie zadanej úlohy. Napríklad ak úloha vyžaduje 4 developerov a 1 analytika, a dvaja developeri ešte pracujú na predošlem úlohe, úloha sa nemôže začať vykonávať, hoci mám k dispozícii väčšinu tímu a reálne by sa na úlohe mohlo začať pracovať. Tzn. *tasky* treba deliť na skutočne elementárne prvky, čiže *sub-tasky*.

3.3 Google OR Tools

3.3.1 Job shop scheduling problem

Google OR Tools podobne ako Python MIP ponúka knižnicu a nie aplikáciu s grafickým rozhraním. Oproti MIP knižnici sú Google OR Tools flexibilnejšie, pretože je možné vybrať si programovací jazyk, v ktorom chcete knižnicu implementovať. Keďže ide o riešenie rovnakého problému ako v kapitole 3.2.1, *Job shop scheduling problem*, a opäť ide o implementáciu knižnice, uvedieme už len hlavné rozdiely medzi MIP a OR Tools, keďže samotné riešenie je veľmi podobné.

OR Tools je explicitnejší pri zápise kódu, čím na jednej strane dodáva väčší priestor model prispôsobiť, na druhej strane je dané riešenie posunuté ešte na nižší level. Počet riadkov na problém *Job shop scheduling* v OR Tools bol 119, počet riadkov na implementáciu rovnakého problému v MIP knižnici bol 58, čo je takmer dvojnásobok. Porovnávame pritom v obidvoch prípadoch impelemntáciu v jazyku Python.

Výhoda oproti implementácii MIP *solvera* je explicitne definovanie stroja a príslušného času potrebného na spracovanie tej-ktorej úlohy. Týmto nie je nutné vkladať nulové hodnoty v prípade, že stroj nemusí danú úlohu spracovať a tým sa vyhneme zbytočnému množstvu zadávaných dát. Toto riešenie zároveň umožňuje riešiť flexibilný *Job shop problem*. Príklad vstupných dát je na Obrázku 12.

Obrázok 12: Formát vstupných údajov v Google OR Tools [15]

```
jobs_data = [  # task = (machine_id, processing_time).
        [(0, 3), (1, 2), (2, 2)],  # Job0
        [(0, 2), (2, 1), (1, 4)],  # Job1
        [(1, 4), (2, 3)]  # Job2
    ]
```

V Prípade MIP musel mať každý zoznam rovnaký počet prvkov, hoci úloha nemusela byť spracovaná na každom stroji. V prípade OR Google Tools je možné definovať dvojice (*machine_id, processing_time*) pre každú úlohu, teda nevznikajú riedke polia a dáta s nulovými časovými intervalmi a každá úloha môže byť spracovaná na strojoch v intervale <1, *m*>, čo v prípade MIP nebolo implicitne podporované.

Tento model teda zodpovedá presnejšie úlohe o priraďovaní zadaných úloh jednotlivým tímom, no stále vyžaduje veľkú réžiu pri spracovaní dát do želanej štruktúry (pole zoznamu *n*-tíc). Príklad formátu dát vytvorených dvojíc je znázornený na Obrázku 13.

Obrázok 13: Príklad vytvorených dvojíc [16]

```
jobs_data = [  # task = (machine_id, processing_time)
            [(3, 13)],
            [(2, 13)],
            [(1, 100)],
            [(1, 20)],
            [(3, 100)],
            [(1, 40)],
            [(1, 8)],
            [(3, 1)],
            [(2, 20)],
            [(1, 5)]
           ]
```

Keďže riešenie nebolo zobrazované priebežne, pravdepodobne tak nebol beh *solvera* spomaľovaný a výsledné optimálne riešenie bolo po spustení nástroja zobrazené

[15] Zdroj: Autor

[16] Zdroj: Autor

takmer okamžite. Riešenie je zreťazené pre jednotlivé stroje vidíme, ako za sebou nasledujú jednotlivé úlohy. Tento spôsob výpisu má väčšiu výpovednú hodnotu, ako výpis riešenia v MIP. Výsledok riešenia bol zobrazený v konzole ako nasledovný text:

Obrázok 14: Výsledné riešenie problému v Google OR Tools [17]

```
Výsledok: Optimálna dĺžka rozvrhu: 173
Stroj 1: job_2_0   job_3_0   job_5_0   job_6_0   job_9_0
         [0,100]   [100,120] [120,160] [160,168] [168,173]
Stroj 2: job_1_0   job_8_0
         [0,13]    [13,33]
Stroj 3: job_0_0   job_4_0   job_7_0
         [0,13]    [13,113]  [113,114]
```

Ďalšia z výhod tohto nástroja je značne jednoduchšia inštalácia. Inštalácia knižnice MIP bola v porovnaní s Google OR Tools náročnejšia, pretože medzi balíčkami vznikali závislosti a niektoré verzie neboli kompatibilné s verziou jazyka Python. Oproti tomu, Google OR Tools vyžadovali iba jednu inštaláciu, ktorá prebehla bez problémov vďaka tomu, že nebolo potrebné spravovať závislosti medzi balíčkami a jednotlivé verzie ako MIP knižnice v Pythone.

Celkovo je teda tento nástroj nepovažujeme za vhodný pre manažérske riešenia, pretože vyžaduje veľkú réžiu pri spracovaní vstupných dát a najmä pri implementácii. Jeho veľkou výhodou však bola rýchlosť a flexibilnosť dátovej štruktúry bez nutnosti zadávať nulové časy pre neexistujúce úlohy.

Týmto sme zhrnuli výhody a nevýhody všetkých testovaných nástrojov a použitých úloh plánovania. Ozrejmili sme bližšie, ako vyzerá práca s týmito nástrojmi a popísali sme, pre ktoré prípady použitia sú vhodné, čím sme splnili čiastkové ciele. Z týchto možností sme vybrali jeden, ktorý sme sa rozhodli ďalej použiť na riešenie problému plánovania projektového portfólia. V nasledujúcej kapitole budeme popisovať aplikáciu vybraného nástroja, detailnejší popis práce s ním a budeme prezentovať jeho výsledky.

[17] Zdroj: Autor

3.4 Vybraný nástroj

Pre výsledné riešenie problému plánovania projektového portfólia sme si zvolili OptaPlanner, vďaka jeho schopnosti komplexne modelovať optimalizačné úlohy. V nasledujúcej časti predstavíme zvolený postup, implementáciu a celkové výsledky optimalizačnej úlohy.

3.4.1 Vstupné dáta a príprava dát

Celková množina dát pozostávala zo 135 úloh exportovaných zo systému JIRA. Do týchto úloh boli zahrnuté všetky úlohy (*Task, Epic, Sub-Task, Story*), ktoré boli v danom čase ohodnotené *Story points*. V Produktovom *Backlogu* sa totiž nachádza väčšie množstvo úloh, tie sú však ohodnocované až počas *Planning Poker* hry, ktorá prebieha pravidelne, nie však nutne po každom *sprinte*. Preto sa odhady v relatívnych jednotkách priradzujú postupne, a teda nie je možné naplánovať vývoj celého produktu od jeho začiatku až po jeho koniec len na základe produktového *backlogu*.

Súčet ohodnotení týchto úloh bol 1932,125 SP. Celkový objem práce, ktorú dokážu spolu všetky tímy za jednu iteráciu vykonať, je 135,25 SP, pričom dĺžka jednej iterácie sú dva týždne. Pri prepočte na priemernú výkonnosť (*velocity*) tímu ((39 + 24,75 + 22 + 18,5 + 17 + 14) / 6) = 22,54 by tieto úlohy mali byť vypracované za približne 14 iterácií, čo v našom prípade znamená 28 týždňov, resp. 7 mesiacov, za predpokladu, že každý tím by mal rovnaké znalosti a skúsenosti so všetkými projektami. Reálna priemerná výkonnosť tímov za jednu iteráciu je uvedená v Tabuľke 2.

Je dôležité spomenúť, že každý tím sa do určitej miery spolupodieľa na riešení chýb v kóde, tzv. *bugfixing*. Keď príde nová chyba, ktorú treba v danej verzii opraviť, tejto chybe nie sú priradené *Story points*. To znamená, že výkonnosť tímu sa týka skutočne iba vývoja nových funkcionalít a s tým súvisiacich aktivít - analýza a testovanie. Do výkonnosti sa teda nezapočítava práca na *bugfixovaní*, teda výkon každého tímu je znížený o túto prácu.

Tabuľka 2: Priemerná výkonnosť tímov za jednu iteráciu

Tím	**Výkonnosť (SP/i)**
Enigma	39
Carmageddon	24,75
Drakarys	21
Lankas	19,5
Voltage	17
Enigma	39

Príbuznosť tímov k projektom

V pôvodnom doménovom modeli boli zamestnanci priradení k zákazníkom. V našom prípade bola implementovaná úprava, vďaka ktorej priraďujeme tímom ich príbuznosť k jednotlivým projektom. Afinitu sme určili nasledovne, pričom v riadkoch sú uvedené hodnoty pre tímy (N - NONE, L - LOW, M - MEDIUM, H - HIGH) a v stĺpcoch sú jednotlivé projekty:

Tabuľka 3: Afinita (príbuznosť) tímov k projektom

Tím/ projekt	**VHR**	**ENR**	**SDK**	**CLN**	**REST**	**CABS**	**RBC**	**MSA**
Voltage	N	L	L	N	L	N	H	N
Cabs	N	L	N	N	N	H	N	N
Carmageddon	H	H	L	L	M	M	L	N
Enigma	M	H	H	H	M	N	N	L
Lankas	L	N	N	N	M	N	N	H
Drakarys	N	N	N	N	M	N	N	H

Zručnosť tímu

Ďalej bolo potrebné vytýčiť zoznam zručností, ktoré majú tímy a typy úloh. Tieto kategórie spolu navzájom úzko súvisia. Zvolili sme preto 6 základných zručností, tak, aby sme ich vedeli k jednotlivým úlohám, ako aj k jednotlivým tímom a 4 základné typy úloh. Aby dávali zmysel, zvolili sme schopnosti pracovať s Androidom a iOS,

ktoré majú všetky tímy, ďalej s *backendom* aplikácie, s *Web View*, manuálmi a konektivitou auta. Matica schopností tímov je uvedená v Tabuľke 4.

Napriek tomu, že každý tím má zručnosť vývoja pre obe mobilné platformy (Android aj iOS) a tieto zručnosti teda nemusia byť zahrnuté do výsledného riešenia, pretože nijako nevplývajú na jeho výsledky, rozhodli sme sa ich ponechať. Po prvé, pre možnú kontrolu výsledkov, po druhé, pre realistickejšiu simuláciu príkladu, po tretie, pre prípad, že sa objaví v *Backlogu* úloha, ktorá by sa týkala iba jednej platformy.

Typy úloh sú rozdelené na všeobecnú prácu na aplikácii, kam reálne spadajú úlohy týkajúce sa vylepšenia grafického dizajnu, práca s formulármi a jednotlivými obrazovkami aplikácie. Ďalej práca s databázami, SDK a *backendom* aplikácie. Ďalší typ úlohy sa týka práce so signálmi a napojením na vozidlo a posledný typ úlohy sa orientuje na preberaní riešení z iných koncernových aplikácií. Zoznam typov úloh je uvedený v Tabuľke 5.

Tabuľka 4: Matica zručností

Tím	**Back End**	**Manuals**	**Android**	**iOS**	**WebView**	**Konektivita**
Voltage	1		1	1		1
Cabs	1		1	1	1	
Carmageddon	1		1	1	1	1
Enigma	1		1	1		
Lankas		1	1	1		
Drakarys		1	1	1		

Tabuľka 5: Zoznam typov úloh

Typ úlohy	**id**
Vývoj na aplikácii	10
Vývoj BE/ DB	12
Vývoj Konektivita	14
Vývoj Koncernové riešenia	16

Následne boli dáta prevedené do XML formátu a každej entite bolo priradené referenčné číslo, podľa ktorého sa na seba jednotlivé entity mohli odkazovať. Úlohám bolo potrebné priradiť počiatočnú dĺžku spracovania, prioritu (LOW, HIGH, MEDIUM) a najskorší možný počiatočný čas spracovania, ktorý bol primárne nastavený na 0. Takto pripravený súbor XML, zobrazený na Obrázku 15, slúžil ako vstupný súbor pre riešenie úlohy *Task assigning problem* v OptaPlanner.

Obrázok 15: Formát vstupných údajov v OptaPlanneri [18]

```
<TaTask id="47">
        <id>1</id>
        <taskType reference="12"/>
        <indexInTaskType>2</indexInTaskType>
        <customer reference="22"/>
        <readyTime>0</readyTime>
        <priority>MAJOR</priority>
        <baseDuration>100</baseDuration>
</TaTask>
```

Výhodou OptaPlanneru je v tomto prípade možnosť použiť nenulový počiatočný čas spracovania, ktorý do istej miery dokáže simulovať poradie, v ktorom tieto úlohy chceme spracovať. Najskorší možný čas spracovania sme nechávali nastavený pre každú úlohu nastavený nulovú hodnotu. V Scrum metodike je tento postup správny. *User Stories*, funkcionality v *Backlogu*, sú vzájomne oddelené a *Tasky* sú zároveň vytvorené ako atomické zadania, ktoré si každý developer môže vybrať z *Backlogu* v ľubovoľnom poradí a pracovať na ich implementácii. V tomto prípade však abstrahujeme od pravidelného plánovania a presúvania úloh z Produktového *Backlogu* do *Sprint Backlogu*, čo v realite jemne koriguje *Scrum Master* a *Product Owner*, a tak zamedzuje tímu vybrať z *Backlogu* akúkoľvek úlohu. Naše riešenie tento fakt však nemení, pretože využívame prioritu každej úlohy, ktorá zabezpečuje, že te pre zákazníka najvýznamnejšie funkcie budú implementované ako prvé.

[18] Zdroj: Autor

Tabuľka 6: Popis XML prepisu dát v tabuľke

id	Task id	Task Type	Index	Reference	Ready	Priority	Pinned	Duration
47	1	12	2	22	0	Medium	False	100
48	2	10	1	19	0	High	False	100
49	3	10	1	19	0	Medium	False	100
50	4	12	2	21	0	Medium	False	80
51	5	16	4	26	0	Medium	False	40
52	6	16	4	26	0	Medium	False	40
53	7	16	4	26	0	Medium	False	40
54	8	16	4	26	0	High	False	40
55	9	12	2	21	0	MEDIUM	FALSE	40
56	10	16	4	24	0	MEDIUM	FALSE	40

3.4.2 Výpočet skóre

Každá trieda, ktorá rozširuje *PlanningSolution* má implementované skóre. Skóre je snaha o objektívne vyjadrenie numerickej hodnoty, ktorá môže porovnať dve riešenia. *Solver* pritom vždy hľadá riešenie s najlepším skóre. Najlepšie možné riešenie má teda najvyššie skóre, ktoré počas riešenia dokázal *solver* nájsť a toto riešenie môže byť optimálne.

Spôsobov, akým sa skóre počíta, je v OptaPlanneri implementovaných niekoľko a je možné vytvoriť si vlastný komparátor. Do výsledného výpočtu sa započítavajú pozitívne a negatívne hodnoty vypočítané na základe podmienok, ktoré musia byť maximalizované alebo minimalizované. Okrem toho je možné do skóre započítať váhu hodnôt - v našom prípade ide o prioritu úlohy (LOW, MEDIUM, HIGH) a úroveň afinity (NONE, LOW, MEDIUM, HIGH). Ďalším vstupom pre výpočet skóre riešenia je úroveň obmedzujúcich podmienok - *hard* a *soft constraints*.

3.4.3 Úprava kódu

V nástroji OptaPlanner sme potrebovali upraviť *Task Assigning* úlohu pre vlastné potreby, pretože pôvodná úloha sa týkala zamestnancov, ktorým boli

priraďované úlohy zákazníkov. Primárne išlo teda o zmeny v doménovom modeli a v grafickom rozhraní, kde bolo potrebné napríklad zobrazovať iterácie namiesto hodinových intervalov a podobne.

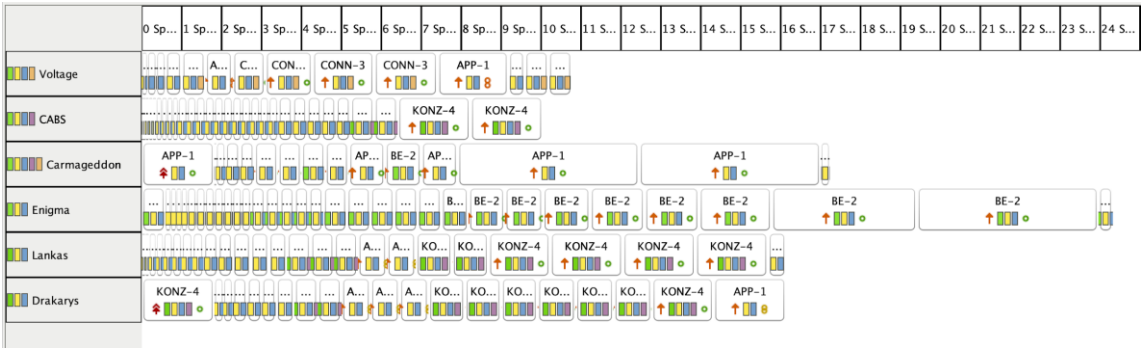
Ďalšia úprava kódu sa týkala grafického rozhrania, kam bolo potrebné pretaviť všetky zmeny doménového modelu. Po prvé išlo o zobrazenie detailov o tímoch a o úlohách pre prechode myšou nad danú entitu. Po druhé išlo o úpravu panelu s časovými intervalmi. Keďže v pôvodnej úlohe boli plánované úlohy pre zamestnancov, a pre každého zamestnanca trvá jedna hodina presne 60 minút, časový panel bol rozdelený do blokov po hodinách a dňoch pre všetkých zamestnancov. V našom prípade sme potrebovali naškálovať jednotlivé úlohy a použiť rovnaké časové jednotky. Keďže objem úloh, resp. tempo iterácie, je pre každý tím iný, napr. pre prvý tím je to 39 *Story points* a pre druhý tým je to 25 *Story points*, ich časové bloky by sa nezhodovali. Preto sme pracovali v blokoch, ktoré reprezentovali priemernú výkonnosť tímu za jednu iteráciu, a to 22 *Story points*.

3.5 Výsledok práce

V tejto kapitole stručne popisujeme výsledky, ktoré sme získali pomocou OptaPlanneru pri plánovaní projektového portfólia. Na nasledovnom obrázku je znázornené výsledné riešenie, ktoré sa pri každom behu ustálilo približne po niekoľkých minútach. V hornej lište je určené, o ktorú iteráciu v poradí sa jedná, pričom ide stále o *sprinty* ktoré určujú objem, nie čas, čiže priemerná iterácia berie do úvahy objem práce s veľkosťou 22 SP. V riadkoch sú priradené tímom jednotlivé úlohy, ktorých šírka reprezentuje reálne dĺžku ich spracovania daným tímom. Dĺžka danej úlohy sa v čase menila, ako bola priraďovaná rôznym tímom, a teda na základe afinity bola vypočítaná aj iná reálna hodnota spracovania úlohy na základe jej pôvodnej *Base Duration*.

Farebne sú odlíšené typy úloh a zručnosti tímov, ktoré sú zobrazené ako na obdĺžnikoch s úlohami v rozvrhu, tak pri názve tímu vľavo.

Obrázok 16: Výsledné riešenie v OptaPlanneri [19]



Podľa *solvera*, ktorý počíta s priemernou výkonnosťou tímu vypočítanou v 3.4.1, dané tímy by skončili vývoj v iterácii č. 10, 9, 17, 24, 15, 15. Tu je však potrebná následná kalibrácia podľa skutočnej výkonnosti tímu, ktorá bola uvedená v Tabuľke 2. Pri prepočte *Story points* podľa priradených úloh tak výsledok pre uniformné tímy s priemernou výkonnosťou 22 SP/ *i* tak je riešenie znázornené na Obrázku 17.

Obrázok 17: Riešenie pre tímy s uniformným priemerným výkonom [20]

Tím/ iterácia	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Voltage																								
Cabs																								
Carmageddon																								
Enigma																								
Lankas																								
Drakarys																								

Ako sme vysvetľovali v kapitole 3.4.1, pôvodné grafické riešenie reprezentovalo jednu iteráciu s objemom práce 22 *Story points*. Keďže nejde o presné časové jednotky a každý tím za jednu iteráciu dokáže vypracovať iné množstvo úloh, s inou výkonnosťou (*velocity*), následne sme museli výsledný počet iterácií prepočítať podľa skutočnej výkonnosti tímov. Korekcia podľa skutočného výkonu tímov je v nasledovnej tabuľke.

[19] Zdroj: Autor

[20] Zdroj: Autor

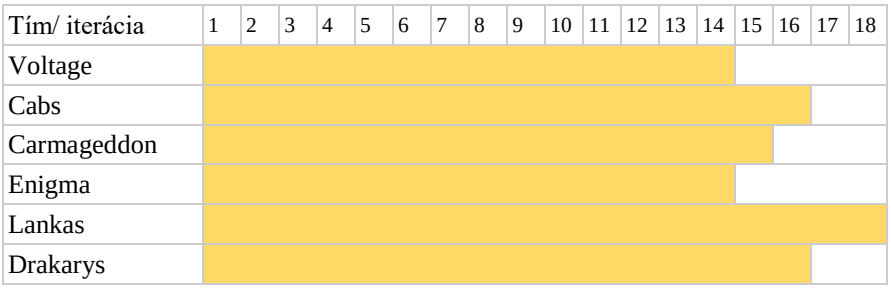
Tabuľka 7: Korekcia výsledkov počtu iterácií

Tím	**Výkonnosť (SP/i)**	**Suma SP**	**Počet iterácií**
Enigma	39	535	13,71794872
Carmageddon	24,75	379	15,31313131
Drakarys	21	355	16,13636364
Lankas	19,5	350	18,91891892
Voltage	17	237	13,94117647
Cabs	14	221	15,78571429

Skutočný počet iterácií, ktoré tímy budú pracovať na im pridelených úlohách, je teda vypočítaný ako súčet objemu pridelených úloh vydelená skutočnou výkonnosťou. Je potrebné dodať, že aj táto skutočná výkonnosť každého tímu bola vypočítaná ako priemerná výkonnosť tímu za posledných *n* iterácií. V každom prípade je však výsledné riešenie najpresnejší možný odhad, aký sme schopní vytvoriť, vzhľadom na existujúce možnosti riešenia a dané vstupy.

Grafické riešenie po korekcii výsledkov na základe reálnej výkonnosti tímov sa nachádza na Obrázku 18. Najväčší rozdiel je vidieť na tíme Enigma, ktorý by s priemernou výkonnosťou dokončil svoje úlohy v 24. iterácii. Keďže ide o najväčší tím, v ktorom sú najskúsenejší developeri, aj výkonnosť tohto tímu je najväčšia (39 SP). Preto sa aj pri objeme priradených úloh 535 SP (čo je najviac spomedzi všetkých tímov) dĺžka vývoja skrátila len na 14 iterácií, čo je v prepočte 28 týždňov.

Obrázok 18: Riešenie pre tímy po korekcii výsledkov [21]



[21] Zdroj: Autor

Výsledné riešenie môžeme ľahko overiť so skúsenosťami, ktoré máme. Z praxe vieme, aké rýchle a aké veľké tímy pracujú na vývoji projektov v tomto portfóliu. Zloženie jednotlivých tímov je nehomogénne, čiže tímy majú rôzny počet vývojárov, členovia tímov majú rôznu senioritu a sú rôzne dlho na projekte. Výsledky, ktoré sme získali, zodpovedajú schopnostiam tímov a je možné ich reálne využiť pri plánovaní vývoja spomínaných projektov.

Takto spracované výsledky bude možné použiť aj pri prezentácii vyššiemu manažmentu pri plánovaní ďalších krokov. Samozrejme budeme musieť zapracovať časovú rezervu na opravu chýb, spájanie vývojových vetiev a testovanie daného riešenia, no takto sme získali realistický odhad, v akom časovom horizonte dokážeme v čo najkratšom možnom čase vypracovať úlohy zadané v Produktovom *backlogu*, čím sme splnili cieľ tejto práce.

Záver

Plánovanie a odhadovanie je náročná úloha, ktorá vyžaduje prácu s veľkým množstvom vstupov. Takého úlohy nie je jednoduché riešiť ani pomocou softvérových nástrojov, pretože spracovanie úloh z reálneho sveta s ich veľkosťou naberá na komplexnosti riešenia. Preto je potrebné dokázať mnoho informácií abstrahovať a zapísať do modelu tak, aby čo najvernejšie modelovali realitu, no zároveň tvorba modelu a hľadanie riešenia netrvalo dlhšie, ako je nutné.

V tejto diplomovej práci sme vyskúšali niekoľko softvérových modelov a nástrojov na rôznej technologickej úrovni. Porovnali sme jednotlivé riešenia, pričom sme sa snažili nájsť to najvhodnejšie, ktoré by pomohlo riešiť zadaný problém plánovania projektového portfólia.

Jednou z nevýhod, ktorá sa týkala všetkých testovaných riešení, je potreba manuálne pripravovať súbory s vstupnými dátami. V našom prípade boli dáta exportované z aplikácie JIRA vo formáte .xlsx, ako list Microsoft Excel. JIRA ponúka na výber niekoľko formátov pre export dát, rovnako ako množinu dát, ktorú je potrebné exportovať. Aj napriek veľkej rozšíriteľnosti JIRA, veľkému množstvu doplnkov a modulov, či robustnosti OptaPlanneru, však neexistuje nástroj, ktorý by potrebné dáta importoval resp. následne konvertoval do požadovaného formátu.

Pre každý zo spomínaných nástrojov sme našli iné vhodné možnosti využitia. Knižnice MIP a Google OR Tools by sme odporúčali využiť ako medzi krok pri vytváraní vlastného, špecifického nástroja pre vlastné potreby. Tieto knižnice je možné implementovať napríklad do vlastných modulov a vlastných softvérových nástrojov. Ďalšiu možnosť využitia vidíme pri riešení akademických, matematických úloh. Keďže spomínané nástroje sú knižnice bez grafického rozhrania, najvhodnejšie sú pre spracovanie najmä číselných vstupov, ktoré budú slúžiť pre ďalšie spracovanie. Tieto nástroje sú vhodné pre prípady použitia, kedy je hľadaná optimálna hodnota, no samotné usporiadanie výstupov nie je až také podstatné.

Vybraný nástroj OptaPlanner sa osvedčil ako manažérsky nástroj. Jeho inštalácia a implementácia nevyžadovala programátorské zručnosti, a to aj vďaka tomu,

že nástroj obsahuje veľké množstvo predpripravených príkladov, ktoré modelujú veľkú množinu úloh z reálneho sveta. Tieto príklady sa tak dajú prispôsobiť vlastným potrebám, a to aj bez nutnosti zasahovať do kódu. Tento kód je však prístupný a v prípade potreby je veľká výhoda mať možnosť model upraviť. Ďalšou výhodou modelu je grafické spracovanie spracovaných dát a animácia počas spracovania úlohy. Celkovo najviac času zaberá príprava a spracovanie dát. Vstupné dáta je potrebné pripraviť a konvertovať na vhodný formát a výstupné dáta je potrebné naškálovať podľa reálnej metriky výkonnosti tímu. OptaPlanner však považujeme za nástroj, ktorý ma veľký potenciál a aj veľké možnosti využitia pri riešení reálnych problémoch plánovania a rozvrhovania.

Zoznam použitej literatúry

[1] BREZINA, Ivan – ČIČKOVÁ, Zuzana – GEŽÍK, Pavel. Sieťová analýza. Bratislava: Vydavateľstvo EKONÓM, 2012. 198 s. ISBN 978-80-225-3503-8.

[2] COCKBURN, Alistar, Agile Software Development: Software Through Development, Pearson Education, 2001. 304 s. ISBN 9780201699692.

[3] JANÁČEK, Jaroslav. Matematické programovanie. Žilina: Žilinská univerzita v Žiline, Edis, 2003. 225 s. ISBN 80-8070-054-0.

[4] OGC (Office of Government Commerce). Managing successful projects with PRINCE2. 5. vydanie. Londýn: TSO, 2009. ISBN 978-011-3310-593.

[5] PMI (Project Management Institute). The Standard for Portfolio Management. 4. vydanie. Pennsylvania: Project Management Institute, 2017. 127 s. ISBN 978-162825-197-5.

[6] PMI (Project Management Institute). A Guide To The Project Management Body Of Knowledge. 5 vydanie. Project Management Institut, 2013. 589 s. ISBN 978-1935589679.

[7] ROYCE, Winston. Managing the Development of Large Software Systems. Proceedings of IEEE WESCON. [online]. Dostupné na internete: < http://www scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf >

[8] SANTOS, Haroldo G. – TOFFOLO, Túlio A.M. Mixed Integer Linear Programming with Python. 2019. 58 s. [online]. Dostupné na internete: < https://buildmedia.readthedocs.org/media/pdf/python-mip/latest/python-mip.pdf >

[9] About OR-Tools. [online]. Dostupné na internete: < https://developers.google.com/optimization/introduction/overview >

[10] CP-SAT Solver. [online]. Dostupné na internete: < https://developers.google.com/optimization/cp/cp_solver >

[11] Epic, User Story, Task. [online]. Dostupné na internete: < https://www.visual-paradigm.com/scrum/theme-epic-user-story-task >

[12] Job Shop vs Flow Shop: Can Robots Work for Both?. [online]. Dostupné na internete: < https://blog.robotiq.com/job-shop-vs-flow-shop-can-robots-work-for-both >

[13] LeSS Framework. [online]. Dostupné na internete: < https://less.works/less/framework/index >

[14] Project Job Scheduling. [online]. Dostupné na internete: < https://docs.jboss.org/drools/release/6.0.0.Final/optaplanner-docs/html_single/index.html#projectJobScheduling >

[15] Role of The Agile Project Manager. [online]. Dostupné na internete: < https://www.virtualprojectconsulting.com/role-of-the-agile-project-manager/ >

[16] OptaPlanner: OptaPlanner User Guide. 2020. 400 s. [online]. Dostupné na internete: < https://docs.optaplanner.org/7.33.0.Final/optaplanner-docs/html_single/index.html >

[17] Optimization Modeling in Python: PuLP, Gurobi, and CPLEX. 2018. [online]. Dostupné na internete: < http://medium.com/opex-analytics/optimization-modeling-in-python-pulp-gurobi-and-cplex-83a62129807a >

[18] Task Assigning Problem. [online]. Dostupné na internete: < https://docs.optaplanner.org/7.37.0.Final/optaplanner-docs/html_single/index.html#taskAssigning >

[19] The Job Shop Problem. [online]. Dostupné na internete: < https://developers.google.com/optimization/scheduling/job_shop >

[20] View and understand the velocity chart [online]. Dostupné na internete: < https://support.atlassian.com/jira-software-cloud/docs/view-and-understand-velocity-chart/ >

[21] What is a product portfolio roadmap? [online]. Dostupné na internete: < https://aha.io/roadmapping/guide/product-roadmap/what-is-a-product-portfolio-roadmap >