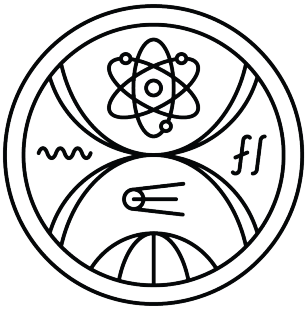
Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

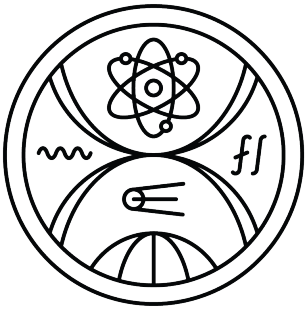


Animácia a vizualizácia diagramov v softvérovom inžinierstve

Diplomová práca

2025 Bc. Karin Kubinová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky



Animácia a vizualizácia diagramov v softvérovom inžinierstve

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. Lukáš Radoský

Bratislava, 2025 Bc. Karin Kubinová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Karin Kubinová
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Animácia a vizualizácia diagramov v softvérovom inžinierstve
Animation and visualisation of software engineering diagrams

Anotácia: Abstrakcia je v softvérovom inžinierstve kľúčovým nástrojom. Softvérové systémy sú dnes značne zložité a táto zložitosť neustále narastá. Najznámejším spôsobom abstrakcie nad softvérovými systémami sú UML diagramy. Hoci v praxi nie je ich notácia striktne dodržiavané, aj spontánne vytvárané diagramy často dodržiavajú vybrané konvencie UML notácie. Diagramy spostredkujú pochopenie systému omnoho rýchlejšie než štúdium zdrojového kódu. Preto má zmysel sa nimi zaoberať, hľadať ich zlepšenia a možné doplnky. Príkladom je animovanie, ktoré statickým UML diagramom dodáva dynamiku a interaktivitu.

Analyzujte existujúce možnosti a prístupy vizualizácie a animácie diagramov v softvérovom inžinierstve. Navrhnite vhodný spôsob zobrazovania a animovania diagramov, napríklad diagramu aktivít v softvéri AnimArch. Implementujte prototyp tohto návrhu. Svoje riešenie overte na vhodne zvolenom príklade vstupného zdrojového kódu, prípadne diagramu, v závislosti od navrhnutého riešenia. Implementáciu overte aj pomocou používateľského testovania. Dosiahnuté výsledky vhodne analyzujte a zhodnoťte.

Cieľ: Návrh, implementácia a evaluácia nového spôsobu vizualizácie a animácie zvoleného typu diagramu v softvérovom inžinierstve

Literatúra:
Yigitbas, E., Gorissen, S., Weidmann, N. et al. Design and evaluation of a collaborative UML modeling environment in virtual reality. Softw Syst Model 22, 1397–1425 (2023). https://doi.org/10.1007/s10270-022-01065-2

Kucecka, J., Vincur, J., Kapec, P., & Cicák, P. (2022). UML-based Live Programming Environment in Virtual Reality. 2022 Working Conference on Software Visualization (VISSOFT), 177-181.

M. Ferenc, I. Polasek and J. Vincur, "Collaborative Modeling and Visualization of Software Systems Using Multidimensional UML", 2017 IEEE Working Conference on Software Visualization (VISSOFT), Shanghai, China, 2017, pp. 99-103, doi: 10.1109/VISSOFT.2017.19.

Kľúčové slová: Softvérové inžinierstvo, Modelovanie softvéru, UML diagram, Vizualizácia, Animácia

Vedúci: Ing. Lukáš Radoský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 08.11.2023

Dátum schválenia: 11.11.2023 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne vyhlasujem, že som túto diplomovú prácu vypracovala samostatne, len s pomocou uvedenej literatúry a pod starostlivým dohľadom vedúceho mojej diplomovej práce.

.....

V Bratislave, 9.5.2025 Bc. Karin Kubinová

Poďakovanie: Ďakujem predovšetkým môjmu školiteľovi Ing. Lukášovi Radoskému za cenné rady, ochotu a odborné vedenie. Ďakujem aj mojej rodine a priateľom za ich podporu a povzbudenie.

Abstrakt

V tejto práci sa zaoberáme analýzou existujúcich prístupov vizualizácie a animácie UML diagramov, s dôrazom na dynamické a interaktívne prvky, ktoré môžu zvýšiť ich efektívnosť. Hlavným cieľom je návrh a implementácia animácie UML diagramu aktivít v softvéri AnimArch.

Implementovali sme vizualizovanie a animovanie diagramu aktivít, ktorý sa generuje z kódu v jazyku OAL, ktorý používateľ zvolí na začiatku animácie. Riešenie sme overili pomocou používateľského testovania, kde účastníci riešili dve úlohy, pričom využívali buď animovaný diagram aktivít v AnimArchu, alebo statický diagram aktivít vytvorený iným nástrojom. Výsledky testovania ukázali, že úspešnosť riešenia úloh sa výraznejšie nelíšila v závislosti od použitého nástroja. Účastníci hodnotili animovaný diagram aktivít pozitívne. Ukázalo sa ale, že by potrebovali viac času na plnohodnotné osvojenie si systému. Z toho môžeme usúdiť, že sa nám podarilo vhodne implementovať vizualizovanie a animovanie diagramu aktivít, ktorý má potenciál prispieť k lepšiemu porozumeniu vykonávaného kódu, keď si používatelia navyknú na tento štýl zobrazovania.

Kľúčové slová: Softvérové inžinierstvo, Modelovanie softvéru, UML diagram, Vizualizácia, Animácia

Abstract

In this work, we analyze existing approaches for visualizing and animating UML diagrams, with an emphasis on dynamic and interactive features that can enhance their effectiveness. The main goal is the design and implementation of UML activity diagram animation in AnimArch software.

We have implemented the visualization and animation of the activity diagram, which is generated from the OAL code that the user selects at the beginning of the animation. We validated the solution through user testing, where participants solved two tasks using either an animated activity diagram in AnimArch or a static activity diagram created by another tool. The results of the testing showed that the success rate of solving the tasks did not vary significantly depending on the tool used. Participants rated the animated activity diagram positively. However, it appeared that they would need more time to fully learn the system. From this we can conclude that we were able to implement visualizing and animating the activity diagram appropriately, which has the potential to contribute to a better understanding of the code being executed once users become accustomed to this style of display.

Keywords: Software engineering, Software modelling, UML diagram, Visualisation, Animation

Obsah

Zoznam obrázkov

Zoznam algoritmov

Terminológia

Skratky

- **CIM** - Model nezávislý od výpočtu, z anglického Computation Independent Model.
- **MASL** - Model Action Specification Language.
- **MDA** - Architektúra riadená modelom, z anglického Model-Driven Architecture.
- **MDD** - Vývoj riadený modelom, z anglického Model-Driven Developmnet.
- **MDE** - Inžinierstvo riadené modelom, z anglického Model-Driven Engineering.
- **MBE** - Inžinierstvo založené na modeloch, z anglického Model-Based Engineering.
- **OAL** - Object Action Language.
- **OMG** - Object Management Group.
- **OMT** - Technika objektového modelovania, z anglického Object Modeling Technique.
- **PIM** - Model nezávislý od platformy, z anglického Platform Independent Model.
- **PSM** - Model špecifický pre platformu, z anglického Platform Specific Model.
- **SADT** - Štruktúrovaná analýza a technika návrhu, z anglického Structured Analysis and Design Technique.
- **SDLC** - Životný cyklus vývoja softvéru, z anglického Software Development Life Cycle.
- **UML** - Unified Modeling Language.
- **xUML** - Spustiteľné UML, z anglického Executable UML.
- **SUS** - System Usability Scale.
- **NASA-TXL** - NASA Task Loader Index.

Úvod

Vizualizácia UML diagramov zohráva v softvérovom inžinierstve kľúčovú úlohu. Softvérové systémy sú často veľmi zložité a ich vnútorné štruktúry nie sú priamo viditeľné. Práve preto využívame vizualizáciu prostredníctvom UML diagramov, aby sme mohli lepšie pochopiť architektúru systému, určiť jeho komponenty, ich vzťahy a správanie. UML diagramy zároveň slúžia ako komunikačný prostriedok medzi analytikmi, vývojármi a ostnatými zainteresovanými stranami, čím sa výrazne uľahčuje návrh, vývoj aj údržba softvéru.

Práve preto sme sa rozhodli implementovať nový diagram do systému AnimArch, ktorý slúži na vizualizáciu a animáciu diagramov. V AnimArchu sa aktuálne animuje triedny a objektový diagram, ktoré reprezentujú štrukturálne aspekty systému. V našej práci implementujeme práve diagram aktivít, aby sme umožnili reprezentáciu aj behaviorálnych vlastností systému. Diagram aktivít budeme v AnimArchu zobrazovať za už existujúcim triednym a objektovým diagramom a budeme ho vytvárať po spustení animácie OAL kódu, ktorý zvolil používateľ. Aktivity sa do diagramu aktivít pridajú pri volaní metódy, pričom budeme aktivity vykresľovať rôzne, podľa typu príkazu, keďže zobrazovanie podmienok a cyklov má v diagrame aktivít zaužívanú konvenciu. Pri volaní novej metódy tiež vykreslíme ďalší diagram aktivít, ktorý budeme zobrazovať za pôvodným diagramom aktivít, čím budeme zobrazovať pomyselný zásobník volaní. Diagram aktivít budeme navyše animovať, čím sa pokúsime používateľovi uľahčiť pochopenie procesov v danom kóde.

Výsledné riešenie budeme evaluovať pomocou používateľského testovania, kedy budeme porovnávať úspešnosť riešenia úloh pomocou nášho animovaného diagramu aktivít a pomocou statického diagramu aktivít. Budeme tiež zisťovať, ako používatelia hodnotili použiteľnosť animovaného diagramu aktivít, v porovnaní so statickým diagramom. Navyše budeme merať aj záťaž, ktorú účastníci testovania vnímali, pri riešení úloh.

V nasledujúcej kapitole 1 sa venujeme modelovaniu softvéru. Opíšeme techniky vývoja softvéru, UML a spustiteľné UML, pričom opíšeme aj jazyk OAL. Zadefinujeme tiež diagram aktivít a jeho základné časti, ktorý budeme v našej práci implementovať. V kapitole 2 opíšeme rôzne nástroje na modelovanie a vizualizovanie softvéru, vrátane systému AnimArch, ktorý rozširujeme o novú funkcionalitu. Základné požiadavky na

rozšírenie systému zadefinujeme a ich implementáciu navrhneme v kapitole 3. Samotnú implementáciu potom opíšeme v kapitole 4. V kapitole 5 navrhneme používateľské testovanie nášho riešenia a vyhodnotíme jeho výsledky.

Kapitola 1

Úvod do problematiky

Medzi neoddeliteľné vlastnosti softvéru patria zložitosť, prispôsobivosť, meniteľnosť a neviditeľnosť [13]. Softvérový systém pozostáva z mnohých častí, ktoré fungujú rôznymi spôsobmi a nelineárne medzi sebou interagujú, čím sa výrazne zvyšuje komplexnosť systému. Keďže softvér nemá vizuálnu reprezentáciu, často prichádza k jeho nepochopeniu, čo môže viesť k rôznym chybám vo vývoji. Práve na to slúži modelovanie softvéru, ktoré si bližšie priblížime v tejto kapitole.

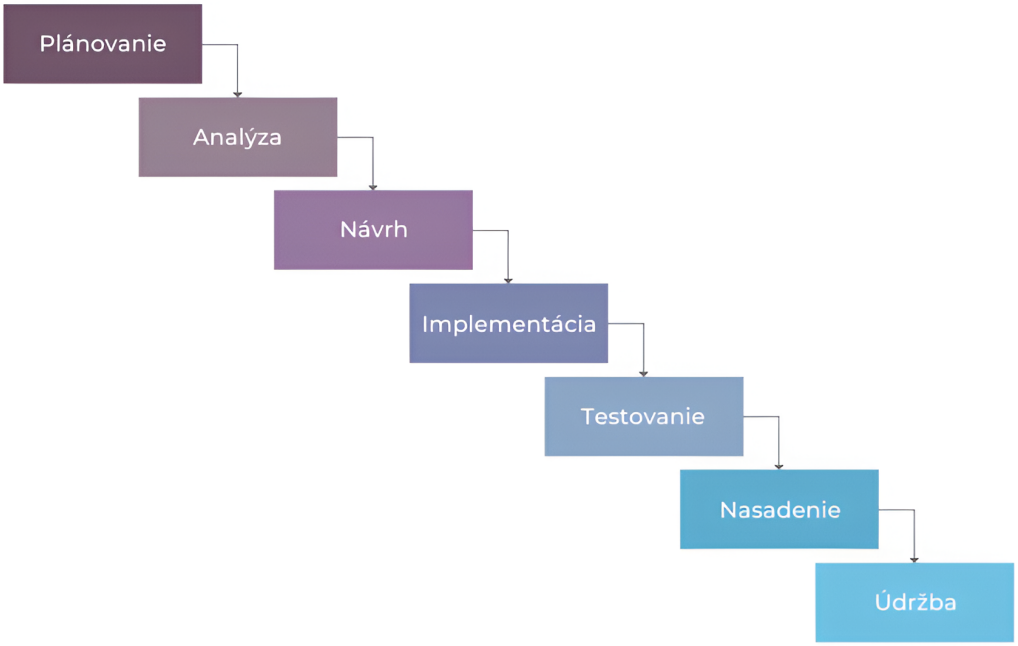
1.1 Modelovanie softvéru

Aby bol proces vývoja softvéru čo najefektívnejší, je vhodné poznať a riadiť sa pravidlami definovanými v SDLC, z anglického Software Development Life Cycle, alebo aj životný cyklus vývoja softvéru. Je to proces jasnej definície požiadaviek, cieľov a fáz tvorby softvéru a zvyčajne pozostáva zo siedmych hlavných etáp [14].

Prvou etapou je *Plánovanie*, kde sa definujú hlavné ciele a požiadavky, identifikujú sa jednotlivé zainteresované strany a vypracuje sa harmonogram. Druhou etapou, ktorá je často spájaná s plánovaním, je *Analýza*. V tejto fáze sa zhromažďujú detailnejšie požiadavky od zainteresovaných strán, na základe ktorých sa potom vytvorí špecifikácia. Ďalej nasleduje *Návrh*, kde sa navrhne štruktúra softvéru, ako budú jednotlivé komponenty vzájomne komunikovať, ale aj celkový dizajn používateľského rozhrania. Po návrhu prichádza samotný *Vývoj* softvéru, ktorý zahŕňa písanie kódu na základe špecifikácii, integrovanie rôznych častí softvéru či jednotkové testovanie jednotlivých komponentov. Piatou etapou je *Testovanie*, kedy sa vykonáva viacero úrovní testovania celého softvéru, odstraňujú sa nájdené chyby a problémy, a taktiež sa overuje splnenie definovaných požiadaviek. Nasleduje *Nasadenie* softvéru do prevádzky a jeho sprístupnenie koncovým používateľom. Poslednou etapou je *Údržba*. Tá trvá počas celého obdobia používania softvéru a zahŕňa aktualizáciu softvéru či opravu chýb a nedostatkov, ktoré objavili používatelia.

Práve vo fázach analýzy a návrhu softvéru sa zvyknú vytvárať rôzne diagramy, ktoré vizualizujú jednotlivé aspekty softvéru a majú za cieľ pomôcť pochopiť komplexné systémy a predísť tak nesprávnej implementácii. Zároveň slúžia ako súčasť dokumentácie, kedy sa nimi môžu vývojári riadiť počas implementácie.

Na obrázku 1.1 vidíme vodopádový model, ktorý je jednou z najzákladnejších SDLC metodológií [14]. Spočíva v lineárnom spôsobe vývoja softvéru, pričom každá fáza procesu musí byť dokončená predtým, ako sa pokračuje na nasledujúcu.



Obr. 1.1: Vodopádový model vývoja softvéru[1].

Proces vytvárania abstraktnej reprezentácie softvéru nazývame *modelovanie softvéru* a slúži na lepšie pochopenie fungovania softvéru, analýzu jeho správania a interakcií medzi jednotlivými komponentmi [15]. Takáto abstraktná reprezentácia sa nazýva model.

1.2 UML

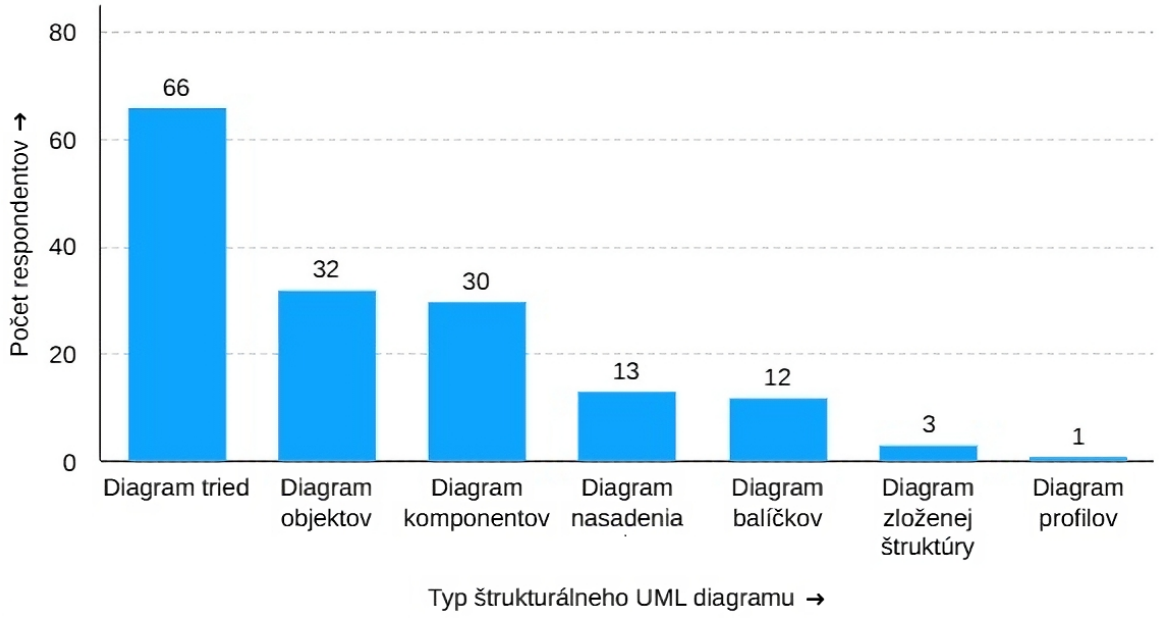
UML, z anglického Unified Modeling Language, je grafický jazyk, ktorý je štandardom pre vizualizáciu, špecifikáciu a dokumentáciu softvéru [16]. Diagram UML je grafická reprezentácia modelu systému a obsahuje UML uzly spojené hranami, ktoré predstavujú prvky v modeli. Tradične sa rozlišujú dva hlavné aspekty systémov a to štrukturálny a behaviorálny aspekt. UML sa tiež riadi týmto delením, a preto sa UML diagramy delia na dva hlavné typy, štrukturálne a behaviorálne [17].

[1] Preložené z `https://www.educba.com/waterfall-model/`.

Štrukturálne UML diagramy popisujú statickú štruktúru systému a jeho častí na viacerých úrovniach abstrakcie. Patrí sem diagram tried, diagram objektov, diagram balíčkov, diagram zloženej štruktúry, diagram komponentov, diagram nasadenia a diagram profilov [2].

Behaviorálne diagramy popisujú správanie systému, a teda ako systém funguje, napríklad prostredníctvom sérií zmien, ktoré sa udejú v systéme za určitý čas. Medzi behaviorálne UML diagramy sa radí diagram prípadov použitia, diagram aktivít a stavový diagram. Sekvenčný diagram, diagram komunikácie, diagram časovania a diagram prehľadu interakcií popisujú interakciu medzi jednotlivými časťami systému a radia sa do podskupiny diagramov interakcií [2].

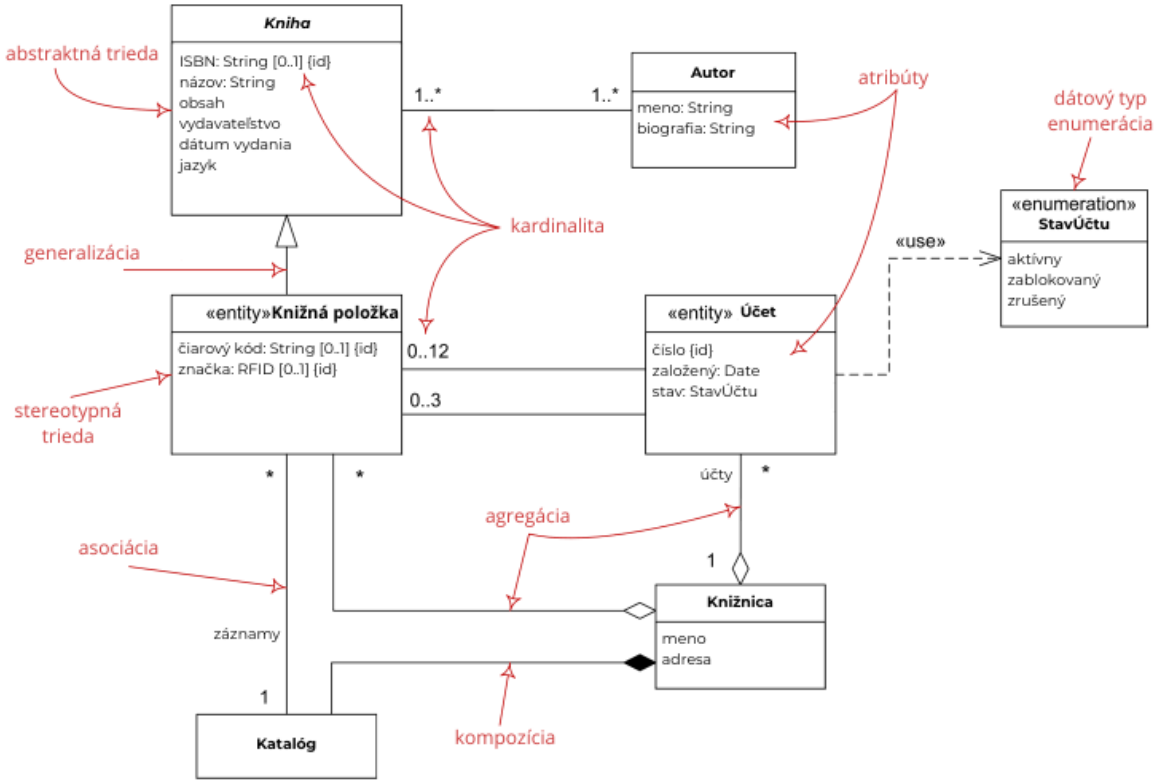
V roku 2016 bola vypracovaná štúdia [1], kedy sa prostredníctvom dotazníku zisťovalo, či sú respondenti oboznámení s Formálnymi metódami, SADT a OMT, či systémoví inžinieri používajú nástroje na správu požiadaviek, ako vytvárajú UML diagramy z požiadaviek, či potrebujú nástroje alebo techniky, ktoré uľahčujú proces od získavania požiadaviek k návrhu softvéru a aké sú podľa nich najpoužívanejšie, respektíve najpotrebnejšie UML diagramy. Dotazník vyplnilo 92 ľudí z akademického alebo IT odvetvia, pričom 85 z nich bolo oboznámených so základnými konceptami UML.



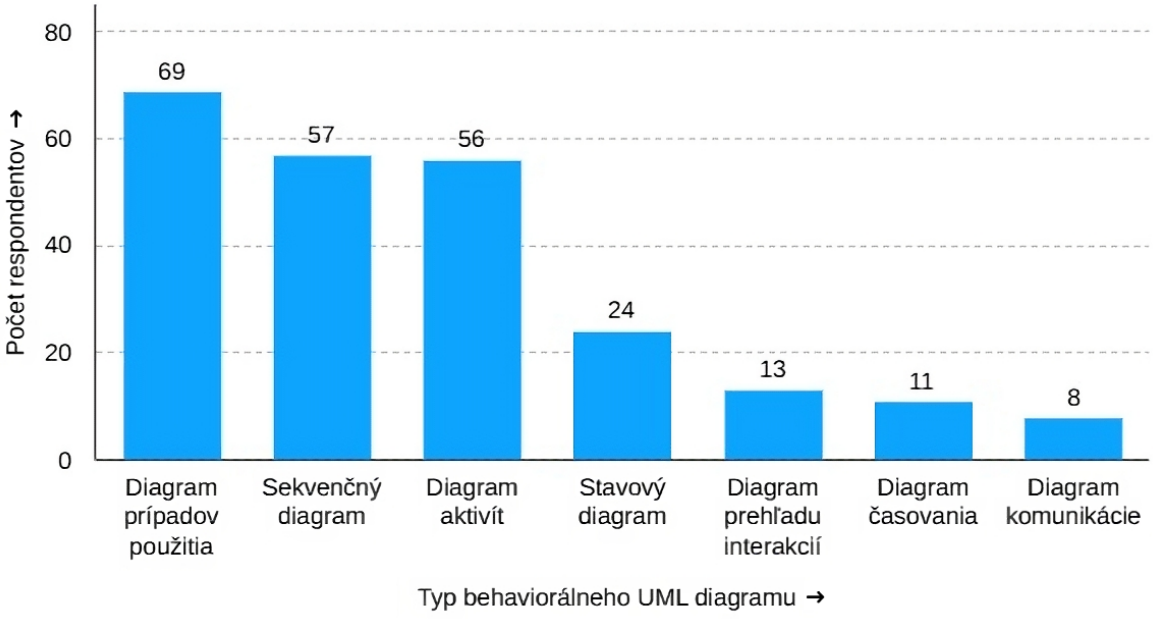
Obr. 1.2: Prehľad použitia štrukturálnych UML diagramov na základe prieskumu, dáta prevzaté z [1].

Na obrázku 1.2 vidíme najčastejšie používané štrukturálne UML diagramy, pričom 66 respondentov určilo ako najčastejšie používaný diagram tried. Ten znázorňuje logický a fyzický návrh systému na úrovni tried a rozhraní a popisuje ich vlastnosti a vzťahy. Príklad diagramu tried vidíme na obrázku 1.3. Sú tu zobrazené rôzne triedy, entity a rozhrania s ich atribútmi. Takisto sú tu znázornené rôzne vzťahy medzi nimi

ako asociácia, generalizácia, agregácia a kompozícia.



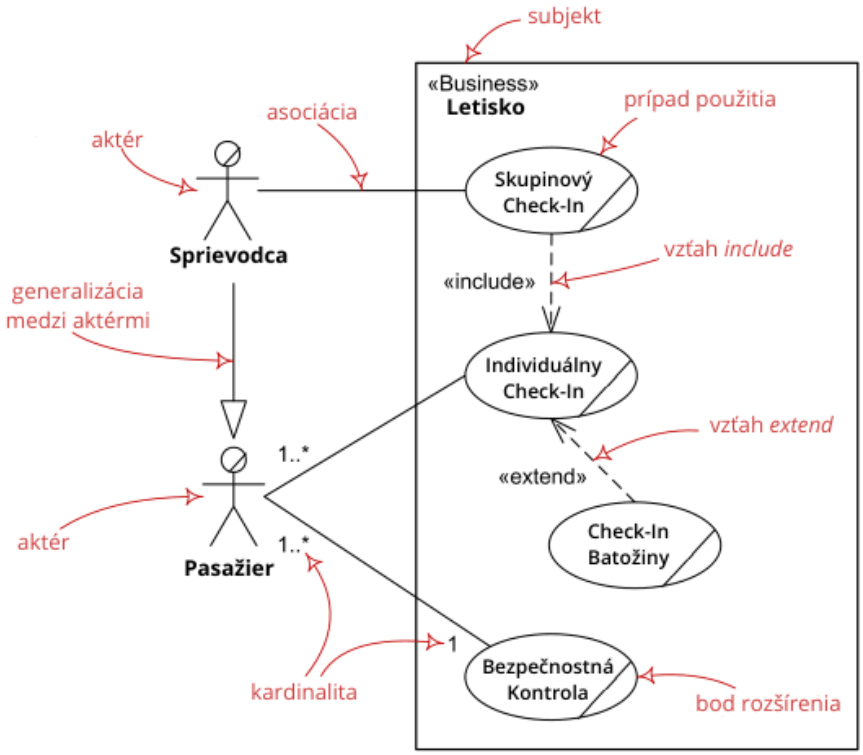
Obr. 1.3: Diagram tried, inšpirované z [2].



Obr. 1.4: Prehľad použitia behaviorálnych UML diagramov na základe prieskumu, dáta prevzaté z [1].

Na obrázku 1.4 sú znázornené najčastejšie používané behaviorálne UML diagramy, pričom 69 respondentov určilo za najpoužívanejší diagram prípadov použitia. Ten zo-

brazuje množinu akcií, prípadov použitia, ktoré sa môžu v systéme vykonať v spolupráci s externými používateľmi systému, aktérmi. Na obrázku 1.5 je príklad takého diagramu prípadov použitia, pričom sú zobrazení dvaja aktéri, ktorí môžu vykonať rôzne akcie. Takisto sú zobrazené vzťahy dedenia medzi aktérmi, asociácie medzi aktérom a prípadom použitia, ale aj vzťah rozšírenia (*extend*) a zahrnutia (*include*) medzi prípadmi použitia.



Obr. 1.5: Diagram prípadov použitia, inšpirované z [2].

1.3 Techniky vývoja softvéru

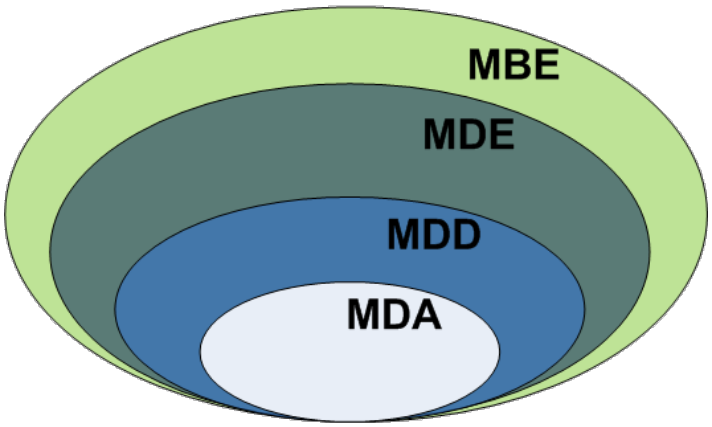
Architektúra riadená modelom, z anglického Model-Driven Architecture (MDA), je prístup, ktorý umožňuje špecifikáciu systémov pomocou modelov. Model je formálna špecifikácia funkcie, štruktúry a správania systému v danom kontexte a zvyčajne sa reprezentuje použitím UML. Architektúra systému je špecifikácia častí a konektorov systému, pričom sú definované aj pravidlá, ako jednotlivé časti interagujú pomocou daných konektorov. V MDA sú tieto časti, konektory a pravidlá definované pomocou modelov [18]. MDA taktiež umožňuje definovanie modelov na rôznych úrovniach abstrakcie. Prvým je model nezávislý od výpočtu, z anglického Computational Independent Model (CIM). Reprezentuje kontext a požiadavky systému, pričom odhliada od štruktúry a technologických detailov systému [18, 19]. Druhým je model nezávislý od platformy, z anglického Platform Independent Model (PIM), pričom pod platformou

sa rozumejú technológie, ktoré poskytujú súbor funkcií, napríklad operačné systémy, programovacie jazyky, databázy a iné [18]. PIM teda špecifikuje systém nezávisle od technológií potrebných na jeho implementáciu. Tretím modelom je model špecifický pre platformu, z anglického Platform Specific Model (PSM), a rozširuje PIM o podrobnosti týkajúce sa používania konkrétnej platformy [18, 19].

Vývoj riadený modelom, z anglického Model-Driven Development (MDD), je technika vývoja, ktorá používa modely ako primárny artefakt vývojového procesu, pričom sa zameriava najmä na fázy požiadaviek, analýzy, návrhu a implementácie [20]. Od MDA sa líši tým, že nedodržiava štandardy vytvorené konzorciom OMG [21], a preto možno chápať MDD ako zovšeobecnenie alebo nadmnožinu MDA.

Inžinierstvo riadené modelom, z anglického Model-Driven Engineering (MDE), je prístup, kedy sú modely kľúčové počas celého inžinierskeho procesu, teda nielen počas vývoja, ale aj počas následnej evolúcie či migrácie softvéru [20], takže opäť možno chápať MDE ako nadmnožinu MDD.

Naopak, ak modely nie sú kľúčovými artefaktmi inžinierskeho procesu, hoci stále zohrávajú dôležitú úlohu, jedná sa o techniku inžinierstva založeného na modeloch, z anglického Model-Based Engineering (MBE), a môže sa vnímať ako nadmnožina MDE [3]. Vzťahy medzi týmito prístupmi sú zobrazené na obrázku 1.6.



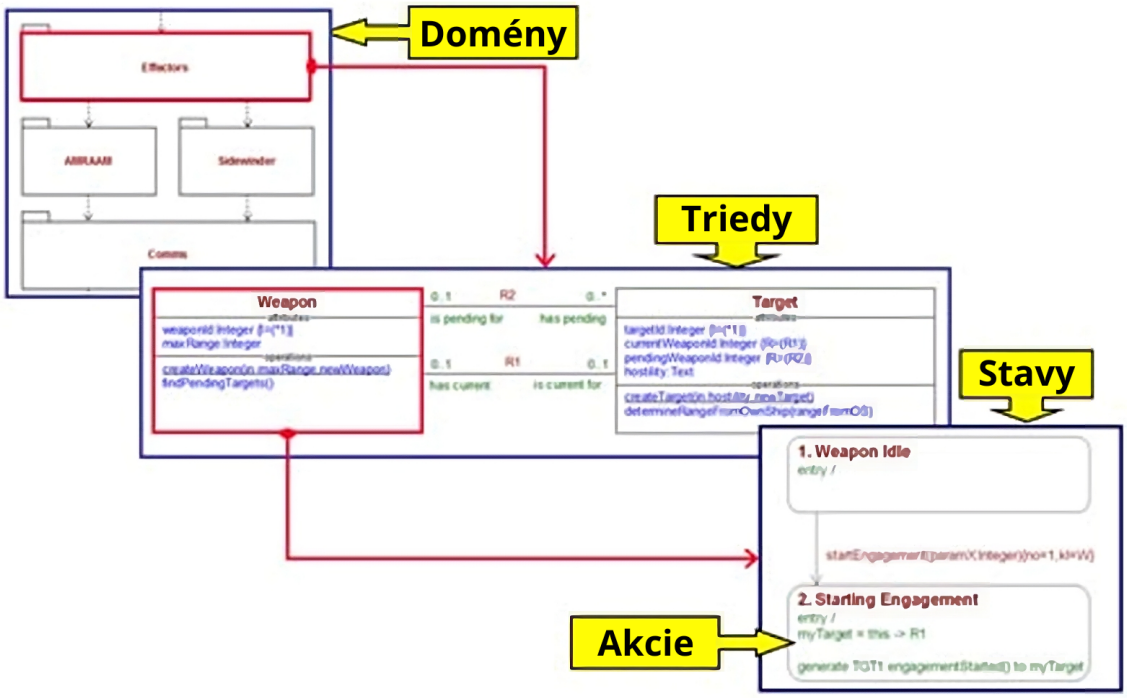
Obr. 1.6: Vzťahy medzi prístupmi MDA, MDD, MDE a MBE, prevzaté z [3].

1.4 Spustiteľné UML

Spustiteľné UML, z anglického Executable UML (xUML), je založené na UML s vyššou úrovňou abstrakcie, pričom odhliada od konkrétnych programovacích jazykov a štruktúry systému, takže špecifikácia vytvorená v xUML môže byť nasadená v rôznych prostrediach bez zmeny [22, 23]. xUML je zároveň jedným z pilierov architektúry riadenej modelom [22, 23], ktorá je opísaná vyššie. Vďaka precíznosti xUML špecifikácie, sa

umožňuje automatická konverzia modelov do programovacieho jazyka na nižšej úrovni abstrakcie, a taktiež ich následné vykonávanie. Pri špecifikácii sa využívajú predovšetkým triedne a stavové UML diagramy, aby bol zabezpečený ako štrukturálny, tak aj behaviorálny aspekt systému, no zvyknú sa využiť aj diagramy aktivít [24], pre lepšie znázornenie procesov v danom systéme.

Základné modelovacie koncepty xUML sú znázornené na obrázku 1.7. Každý systém je najskôr rozdelený do **domén**, pričom tie sú ďalej rozdelené do **tried**. Každá trieda môže mať svoj **stavový stroj**, ktorý spracúva asynchrónne signály vykonávaním **stavových akcií**, a tiež operácie, ktoré spracúvajú synchrónne signály [4].



Obr. 1.7: Základné koncepty xUML, preložené z [4].

Aby boli UML modely spúšťateľné, je potrebné systém ešte bližšie špecifikovať, a to definovaním jednotlivých systémových závislostí od akcií a času, kedy sa majú jednotlivé zmeny vykonať [23]. Na to slúžia jazyky akcií, z anglického Action Languages, ktoré umožňujú zachovanie vysokej úrovne abstrakcie modelov, a teda aj ich následný preklad do spustiteľného kódu.

1.4.1 OAL

Object Action Language (OAL) je jedným z existujúcich jazykov akcií a definuje sémantiku spracovania, ktoré prebieha počas akcie [12]. Pod akciou sa rozumejú modelované prvky ako stavy, funkcie, triedne operácie a iné. Každá akcia sa pritom skladá z viacerých príkazov, či už jednoduchších, ako je prístup k atribútom triedy, alebo zložitejších,

ako napríklad podmienky alebo cykly [12]. Syntax jazyka OAL sa vyznačuje rozlišovaním veľkosti písma a písaním bodkočiarky na konci každého príkazu. Na ukážke kódu 1.1 vidíme príklad podmienky v jazyku OAL, kedy sa priradí do premennej **x** rôzne číslo, na základe mena v premennej **meno**. Na ukážke kódu 1.2 je príklad písania cyklu. Keďže príkazy v cykle môžu byť vykonávané aj paralelne, odporúča sa využívať *foreach* cykly, ktoré iterujú cez všetky prvky v kolekcii, namiesto cyklov s iterovaním cez premennú [12].

```
if (meno == "John")
    x = 1;
elif (meno == "Bill")
    x = 2;
elif (meno == "Michael")
    x = 3;
else
    x = 4;
end if;
```

Alg. 1.1: Ukážka písania podmienky v OAL, prevzaté z [12].

```
// D je klucove pssmeno pre objekt typu Dieta.
// deti je implicitne typovana premenna obsahujuca mnozinu
   instancii <instance handle set> typu D.
select many deti from instances of D;
for each dieta in deti
    generate D1:'cas ist spat' () to dieta;
end for;
```

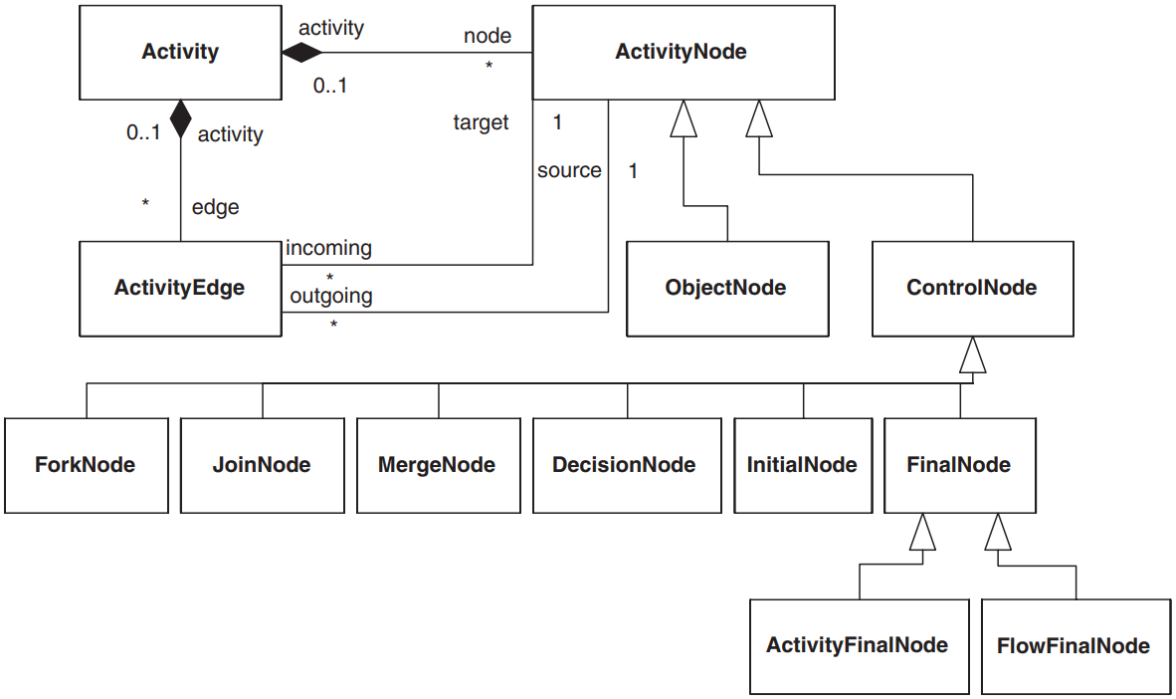
Alg. 1.2: Ukážka písania *foreach* cyklu v OAL, prevzaté z [12].

Medzi vývojové prostredia, ktoré slúžia na návrh, simuláciu a generovanie kódu z xUML patrí napríklad platforma BridgePoint [25], ktorá je založená na vývojovom prostredí Eclipse. BridgePoint podporuje dva jazyky akcií, medzi ktoré patrí už spomenutý Object Action Language (OAL) a Model Action Specification Language (MASL) [26].

1.5 Diagram aktivít

Diagram aktivít je typ behaviorálneho UML diagramu, ktorý zachytáva dynamiku systému, pracovné toky, z anglického *workflows*, a výpočtové, ale aj organizačné procesy systému [27, 6]. Taktiež umožňuje zobrazovať aj procesy, ktoré sa vykonávajú para-

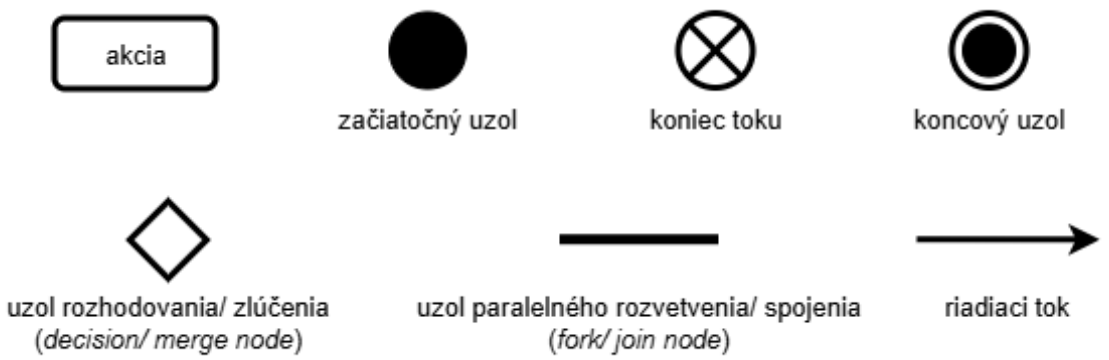
lelne. V špecifikácii UML 1.x boli diagramy aktivít vnímané ako špeciálny typ stavových diagramov, čo ale neumožňovalo plnohodnotné modelovanie pracovných tokov. To sa zmenilo vo verzii UML 2.0, čím sa umožnilo lepšie modelovanie komplexnejších procesov [2, 6]. Pre lepšie pochopenie štruktúry diagramu aktivít slúži metamodel zobrazený na obrázku 1.8. Centrálnym prvkom je metatrieda **`Activity`**, ktorá sa skladá z triedy **`ActivityNode`**, ktorá predstavuje jednotlivé uzly v diagrame, ktoré sa využívajú na znázornenie procesov, a triedy **`ActivityEdge`**, ktorá slúži na zobrazenie prechodov medzi uzlami. **`ObjectNode`** predstavuje objektový uzol, abstraktný uzol aktivity, ktorý sa používa na definovanie toku dát v aktivite. **`ControlNode`** predstavuje riadiaci uzol, ktorý slúži na usmernenie toku medzi ostatnými uzlami. Medzi riadiace uzly patria podtriedy **`InitialNode`**, **`DecisionNode`**, **`MergeNode`**, **`JoinNode`**, **`ForkNode`** a **`FinalNode`**, ktorá sa ďalej špecifikuje na triedy **`ActivityFinalNode`** a **`FlowFinalNode`**. Tieto triedy predstavujú rôzne typy uzlov, ktoré sa môžu nachádzať v diagrame aktivít, pričom sú bližšie špecifikované nižšie.



Obr. 1.8: Metamodel diagramu aktivít, prevzaté z [5].

Diagram aktivít sa skladá z viacerých základých častí, ktoré môžeme vidieť na obrázku 1.9.

Aktivita špecifikuje vykonávanie akcií pomocou toku riadenia [27]. Pod akciou sa teda rozumie jeden atomický krok aktivity. Akcie môžu byť vyjadrené aj jazykom akcií, vďaka čomu sú jediným vykonateľným vrcholom UML [2], čo umožňuje spúšťať ďalšie akcie, pristupovať k objektom a meniť objekty, alebo ich spájať a vytvárať tak zložitejšie akcie. Akcie bývajú znázornené obdĺžnikmi so zaoblenými rohmi.



Obr. 1.9: Notácia základných častí diagramu aktivít.

Začiatočný uzol, znázorňovaný čiernym bodom, predstavuje začiatok toku, keď je aktivita vyvolaná. Aktivita môže mať viacero začiatočných uzlov, v takom prípade každý z nich spustí samostatný tok [2].

Koniec toku ukončuje tok aktivity, pričom ale nemá žiadny efekt na ostatné toky v aktivite [2]. Reprezentuje sa kruhom so symbolom "X" uprostred.

Koncový uzol sa znázorňuje kruhom s čiernym bodom uprostred a ukončuje všetky toky v aktivite.

Uzol rozhodovania, z anglického *decision node*, označuje miesto, kedy sa tok rozdeľuje. Má jednu vstupnú hranu a niekoľko výstupných hrán, pričom výstupná hrana sa určí na základe toho, ktorá z navzájom sa vylučujúcich podmienok bude splnená [2]. Podmienka býva znázornená v hranatých zátvorkách pri uzli rozhodovania alebo pri hrane, ku ktorej patrí. Uzol rozhodovania sa označuje kosoštvorcom.

Uzol zlúčenia, z anglického *merge node*, sa taktiež označuje kosoštvorcom a označuje miesto, kedy sa toky spájajú. Má teda viacero vstupných hrán a jednu výstupnú hranu.

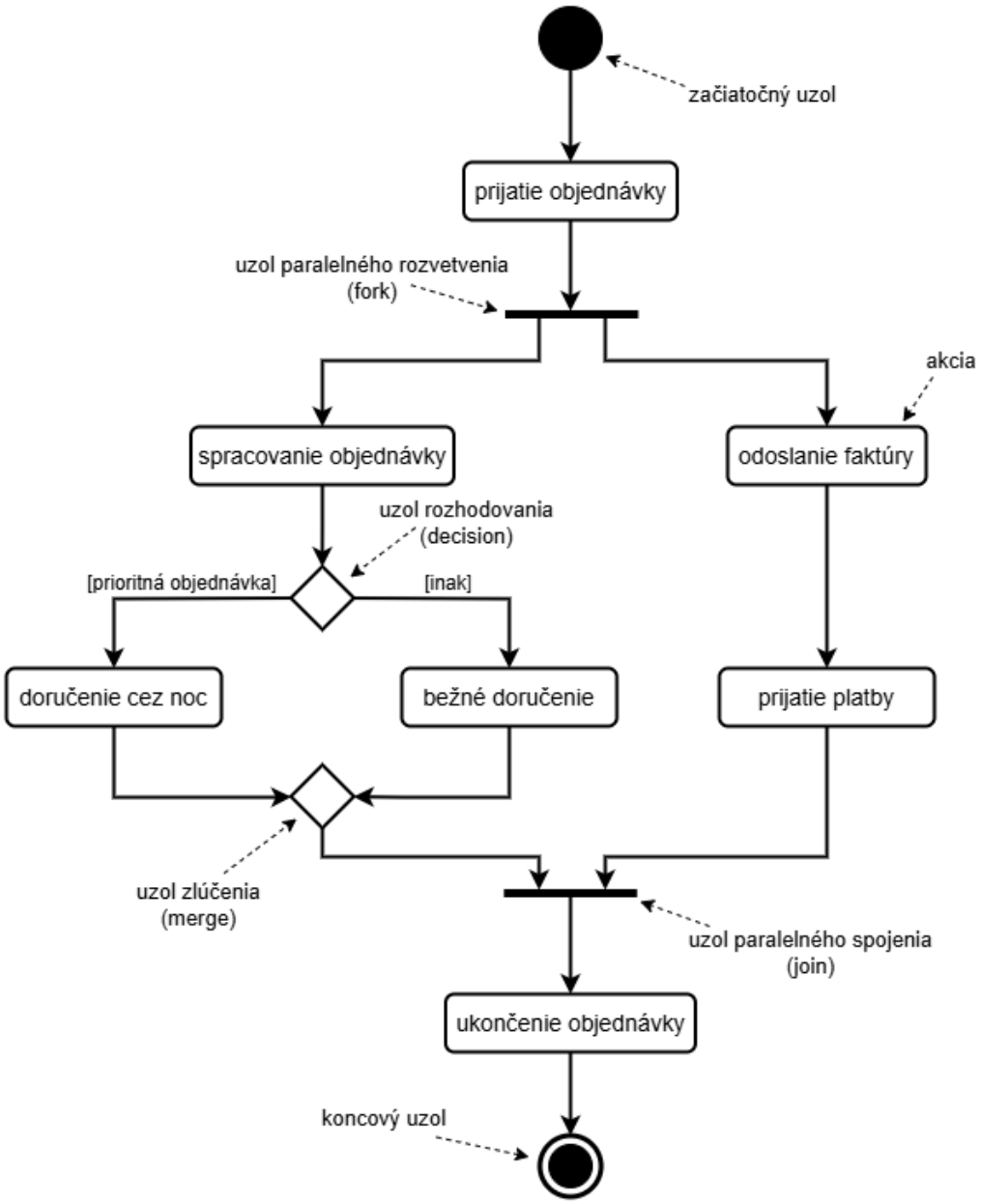
Uzol paralelného rozvetvenia, z anglického *fork node*, predstavuje rozvetvenie toku do viacerých súbežných tokov, ktoré sa budú vykonávať paralelne [2]. Znázorňuje sa hrubšou čiarou.

Uzol paralelného spojenia, z anglického *join node*, označuje synchronizáciu súbežných tokov a taktiež sa znázorňuje hrubšou čiarou.

Riadiaci tok predstavuje chod riadenia medzi jednotlivými akciami a označuje sa šípkami.

Na obrázku 1.10 sa nachádza jednoduchý príklad diagramu aktivít. Vykonávanie sa začne v začiatočnom uzli. Následne sa vykoná akcia *prijatie objednávky*. Po jej vykonaní sa tok rozdvojí v uzle paralelného rozvetvenia do dvoch paralelných tokov, čo znamená, že akcie *prijatie objednávky* a *odoslanie faktúry* (a nasledujúce) sa môžu vykonať v ľubovoľnom poradí, prípadne aj súčasne. Napríklad, začne sa plniť objednávka, potom sa odošle faktúra, doručí sa objednávka a nakoniec sa príjme platba, alebo sa

platba príjme počas doručovania tovaru. Postupnosť akcií medzi paralelnými tokmi je irelevantná, pričom ale v jednotlivom paralelnom toku sa akcie naďalej vykonávajú postupne. Keď sú oba paralelné toky dokončené, opäť sa synchronizujú v uzle paralelného spojenia a vykonávanie pokračuje akciou *ukončenie objednávky*. Po jej dokončení sa tok ukončí koncovým uzlom. Na diagrame je taktiež znázornená podmienka, kedy sa po vykonaní akcie *spracovanie objednávky* tok rozdelí rozhodovacím uzlom, a podľa priority objednávky sa buď vykoná akcia *doručenie cez noc*, alebo *bežné doručenie*. Vetva, ktorá nespĺňa podmienku, sa označuje pojmom [inak], resp. po anglicky [*else*]. Po vykonaní príslušnej vetvy sa toky opäť spoja v uzle zlúčenia.



Obr. 1.10: Príklad diagramu aktivít, inšpirované z [6].

Kapitola 2

Nástroje na vizualizovanie softvéru

V tejto kapitole sú opísané rôzne nástroje na modelovanie a vizualizovanie softvéru.

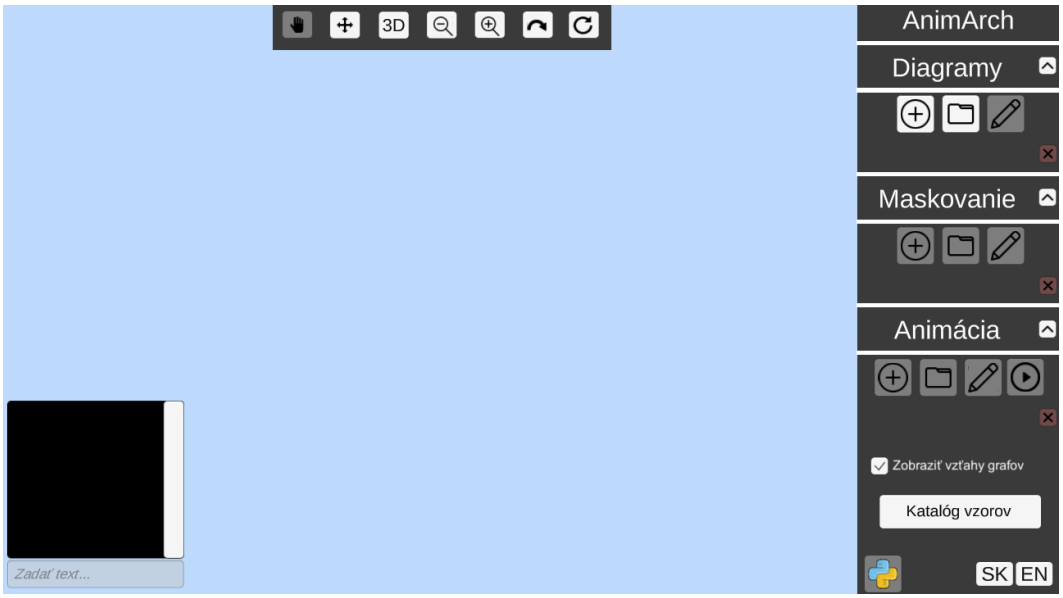
2.1 AnimArch

Keďže cieľom tejto práce je implementovať zobrazovanie a animovanie diagramu aktivít v softvéri AnimArch, je najskôr potrebné sa s ním oboznámiť. AnimArch je komplexný nástroj vyvinutý v Unity použitím jazyka C#. Zaoberá sa dvoma hlavnými oblasťami výskumu, a to vizualizáciou a animáciou UML diagramov a generovaním zdrojového kódu z kombinácie UML diagramu a skriptu v jazyku OAL [28]. Na obrázku 2.1 je zobrazená domovská obrazovka nástroja AnimArch.

AnimArch umožňuje vytvoriť, editovať a načítať triedny diagram, ktorý sa následne vykreslí do 3D priestoru. Po načítaní diagramu je možné ho aj animovať pomocou animácií definovaných v jazyku OAL. Po načítaní a spustení animácie sa taktiež vytvára diagram objektov, ktorý sa nachádza v priestore za triednym diagramom a zobrazuje objekty vytvárané počas vykonávania OAL kódu. Po načítaní triedneho diagramu a animácie s OAL kódom je taktiež možné vygenerovať spúšťateľný kód v jazyku Python, čím sa uplatňuje technika vývoja MDD.

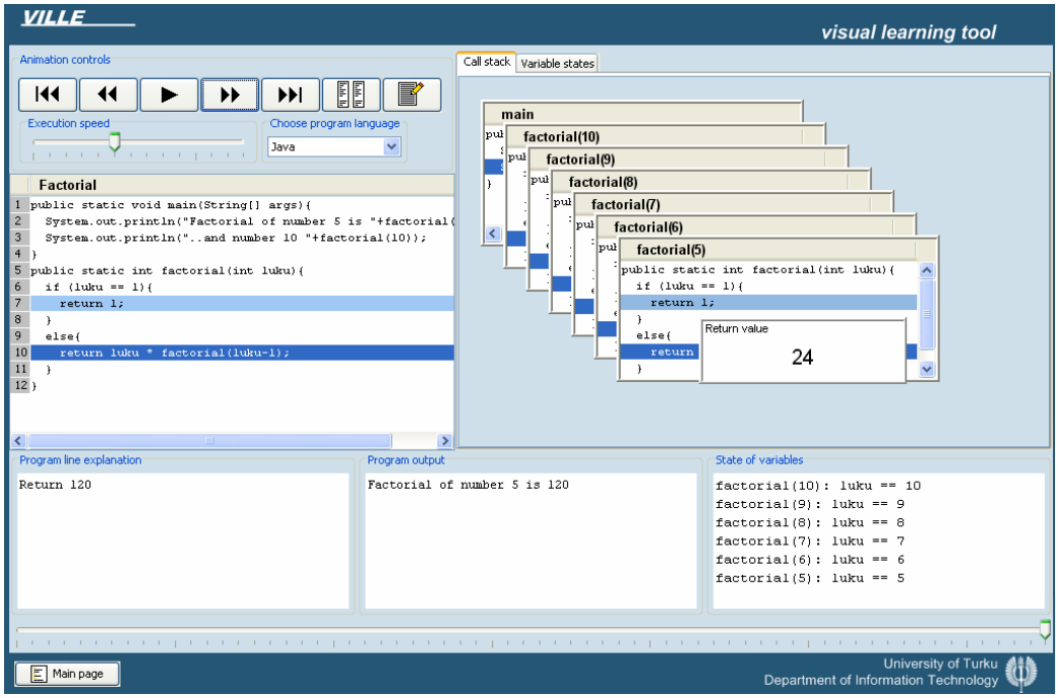
2.2 VILLE

Ďalším vizualizačným nástrojom je jazykovo nezávislý nástroj VILLE, ktorý umožňuje vytváranie a editovanie programovacích príkladov v rôznych programovacích jazykoch [7]. Takisto umožňuje pozorovanie rôznych udalostí počas vykonávania týchto príkladov. Pre každý riadok kódu sa tiež automaticky generuje slovné vysvetlenie toho, čo daný príkaz robí, čo môže byť užitočné najmä pri výučbe. Počas vykonávania kódu je používateľovi tiež umožnené pohybovať sa medzi príkazmi o krok vpred, či vzad, čo uľahčuje používateľovi pochopenie vykonávaných príkazov. Počas vykonávania sa



Obr. 2.1: Domovská obrazovka softvéru AnimArch.

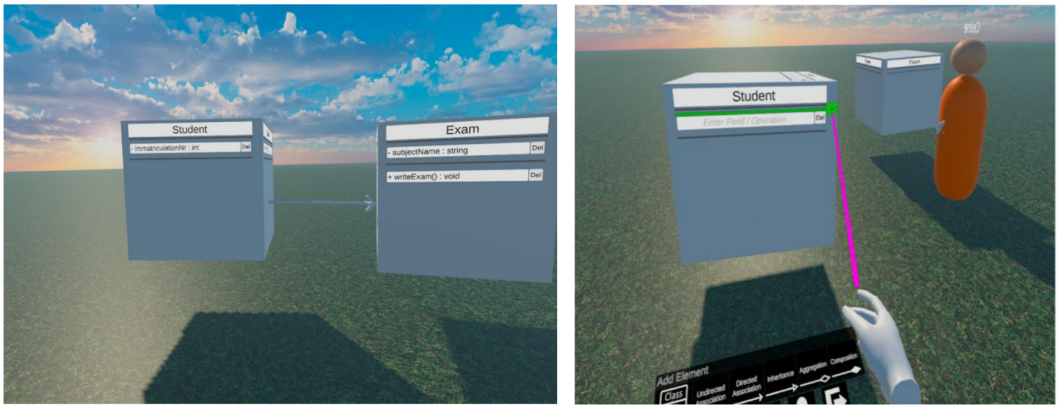
zobrazuje aj zásobník volaní, ktorý môžeme vidieť na obrázku 2.2, kde sa zobrazujú jednotlivé volania metód a ich návratové hodnoty. Tento prístup je inšpiráciou pre túto diplomovú prácu, avšak s tým rozdielom, že v našej práci budeme jednotlivé príkazy vizualizovať pomocou diagramu aktivít.



Obr. 2.2: Zobrazenie zásobníku volaní vo VILLE, prevzaté z [7].

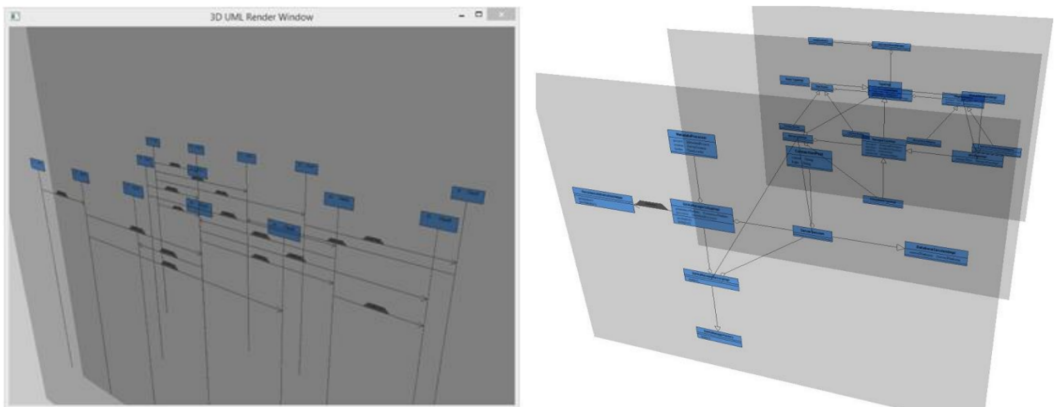
2.3 Iné nástroje

E. Yigitbas a kol. vytvorili modelovacie prostredie vo virtuálnej realite, ktoré slúži na kolaboratívne modelovanie UML triednych diagramov [8]. Potenciál tohto prostredia otestovali s 24 účastníkmi a zistili, že modelovanie vo virtuálnej realite bolo pre používateľov síce menej efektívne, no mali pocit, že sa nachádzajú v tej istej miestnosti ako ich spolupracovníci, čo zároveň dopomohlo k prirodzenejšej spolupráci. Ukážka kolaboratívneho modelovania v tomto prostredí je zobrazená na obrázku 2.3.



Obr. 2.3: Ukážka kolaboratívneho modelovania triedneho diagramu, prevzaté z [8].

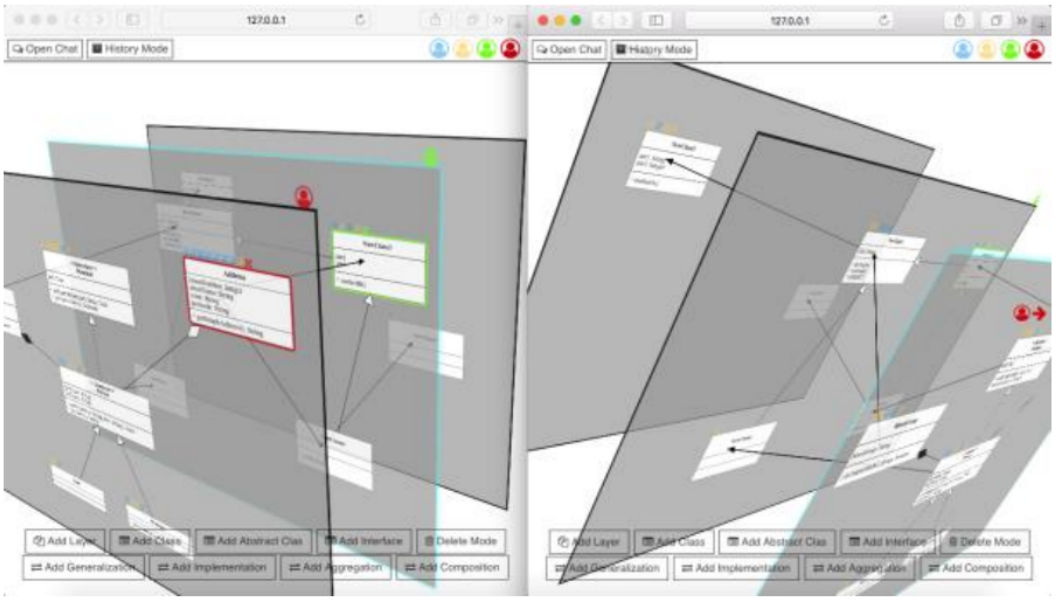
L. Gregorovič a I. Polášek navrhli prístup vizualizovania UML diagramov v 3D priestore, a to prostredníctvom automatického generovania objektových a triednych diagramov zo sekvenčných diagramov [9]. Diagram objektov bol automaticky odvodený zo sekvenčného diagramu a diagram tried sa následne vytvoril z objektového diagramu, kde asociácie sa odvodili z interakcií v sekvenčnom diagrame a metódy tried sa extrahovali z požadovaných operácií v týchto interakciách. Vďaka tomu, že sa diagramy zobrazovali v 3D priestore bolo tiež možné zobraziť viacero diagramov za sebou v rôznych vrstvách, ako je zobrazené na obrázku 2.4.



Obr. 2.4: Ukážka zobrazenia UML diagramov, prevzaté z [9].

I. Polášek spolu s M. Ferencom a J. Vincúrom nadviazali na predošlú prácu, kde

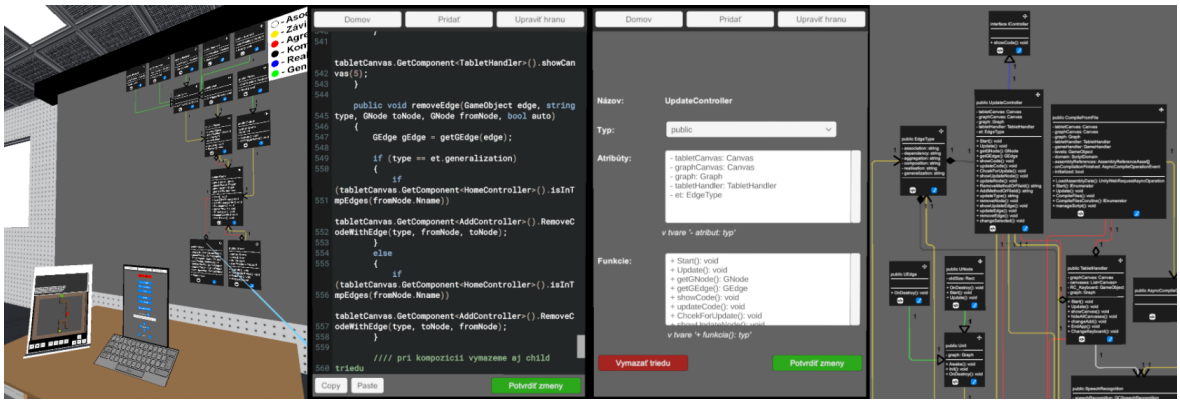
k viacvrstvovým diagramom pridali možnosť synchronizovanej kolaborácie a komunikácie medzi používateľmi v reálnom čase [10]. Systém teda vizualizovali pomocou 2D UML diagramov na prepojených vrstvách v 3D priestore. Do aplikácie implementovali rôzne funkcie, ako oznámenie o prihlásení nového používateľa do aplikácie, evidenciu histórie úkonov používateľa, či časovú os histórie projektu. Taktiež implementovali aj funkciu četu medzi používateľmi. Cieľom týchto funkcií bolo zlepšiť spoluprácu, zvýšiť efektivitu práce a minimalizovať potrebu komunikácie medzi používateľmi. Ukážka tejto aplikácie je zobrazená na obrázku 2.5.



Obr. 2.5: Ukážka kolaboratívnej 3D aplikácie, prevzaté z [10].

J. Kučečka a kol. vytvorili prostredie vo virtuálnej realite (VR) pre kolaboratívne programovanie, ktoré je zobrazené na obrázku 2.6. Umožňuje vývojárom písať, generovať, upravovať a spúšťať zdrojový kód [11]. Zároveň umožňuje, aby sa zmeny vykonané v UML diagrame premietli do zdrojového kódu a naopak. Navrhnuté prostredie tiež evaluovali pomocou štúdie s 20 účastníkmi, pričom sa sústredili najmä na to, či sa môže vývoj softvéru vo VR rovnať klasickému vývoju na 2D monitore, či môže vývoj vo VR zlepšiť používateľov zážitok z vývoja a či môže vývoj softvéru vo VR skrátiť čas potrebný na kódovanie počítačových programov. Zistilo sa, že z hľadiska používateľnosti a atraktívnosti účastníci lepšie ohodnotili VR IDE. Avšak, čo sa týka rýchlosti dokončenia zadania, to trvalo vo VR IDE o niečo dlhšie. Používatelia tiež navrhli určité zlepšenia, napr. zavedenie VR klávesnice s vyšším rozlíšením, keďže súčasná implementácia využívajúca sledovanie klávesnice mala určité obmedzenia.

Väčšina spomenutých nástrojov využívala pri vizualizovaní a modelovaní softvéru virtuálnu realitu. Nad jej implementáciou do nástroja AnimArch, ktorý v tejto práci rozširujeme o novú funkciu, sa ale momentálne neuvažuje, aj kvôli menšej efektívnosti,



Obr. 2.6: Ukážka prostredia na kolaboratívne programovanie vo VR, prevzaté z [11].

ktorú používatelia vykazovali oproti klasickému prístupu. Môžeme sa ale inšpirovať práve spôsobom evaluácie, ktorá sa využila na analyzovanie daných nástrojov, počas hodnotenia našej výslednej práce.

Kapitola 3

Požiadavky a návrh rozšírenia aplikácie

V tejto kapitole zadefinujeme ciele a požiadavky na rozšírenie predošlej aplikácie a navrhneme ich postup implementácie.

3.1 Požiadavky

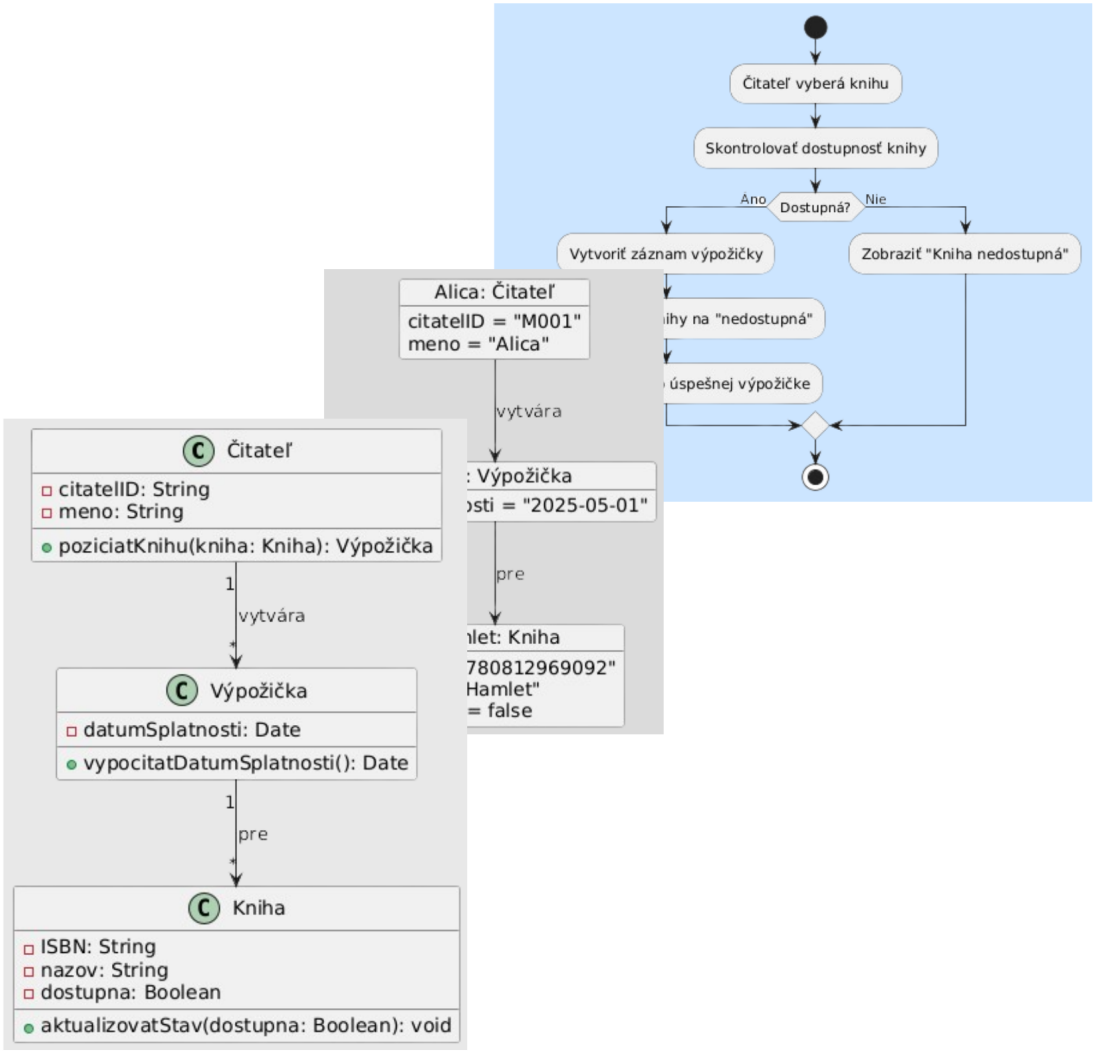
Cieľom našej práce je navrhnúť, implementovať a evaluovať nový spôsob vizualizácie a animácie diagramu aktivít v softvéri AnimArch. V AnimArchu sa aktuálne vizualizujú a animujú diagramy tried a objektov, teda len štrukturálne diagramy, preto sme sa rozhodli implementovať práve diagram aktivít, ktorý bude reprezentovať aj behaviorálne vlastnosti načítaného kódu. Zároveň sme sa tiež inšpirovali nástrojom VILLE, bližšie opísaným v sekcii 2.2. V ňom sa počas vykonávania kódu zobrazuje zásobník volaní, kde sa zobrazujú volané metódy a ich návratové hodnoty, čo v našom prípade budeme zobrazovať pomocou diagramu aktivít. Samotnú implementáciu budeme následne overovať aj pomocou používateľského testovania.

Medzi základné požiadavky na naše rozšírenie AnimArchu teda patrí:

- vytvoriť novú vrstvu na zobrazovanie diagramu aktivít,
- implementovať vytváranie diagramu aktivít po spustení animácie,
- implementovať zobrazovanie viacerých diagramov aktivít za sebou,
- animovať diagram aktivít počas behu animácie,
- evaluovať vytvorené riešenie.

3.2 Návrh aplikácie

V softvéri AnimArch sú aktuálne dve vrstvy diagramov. V prvej vrstve sa zobrazuje diagram tried a v druhej vrstve, ktorá je umiestnená za prvou vrstvou, sa zobrazuje diagram objektov. Diagram aktivít budeme zobrazovať v tretej vrstve za diagramom objektov. Na obrázku 3.1 je zobrazená ukážka návrhu zobrazenia rôznych vrstiev pre diagram tried, diagram objektov a zvýraznená je práve vrstva pre diagram aktivít, ktorú budeme implementovať.



Obr. 3.1: Ukážka návrhu zobrazenia rôznych vrstiev diagramov. Prvá vrstva pre diagram tried a druhá vrstva pre diagram objektov sa v AnimArchu už zobrazujú. Zobrazenie zvýraznenej tretej vrstvy pre diagram aktivít budeme implementovať v našej práci.

Proces vytvárania diagramov v AnimArchu je zobrazený na nasledujúcom sekvenčnom diagrame 3.2, pričom zvýraznené hrubým sú procesy, ktoré implementujeme v našej práci. Spustením AnimArchu sa získajú referencie na diagram tried a diagram ob-

jektov z Unity scény. Tu pridáme tiež získavanie referencie na diagram aktivít, ktorého generovanie budeme implementovať. Používateľ ďalej zvolí súbor triedneho diagramu, ktorý sa vygeneruje a zobrazí. Taktiež k nemu môže vybrať animáciu, ktorá obsahuje kód v jazyku OAL, ktorý sa bude animovať.

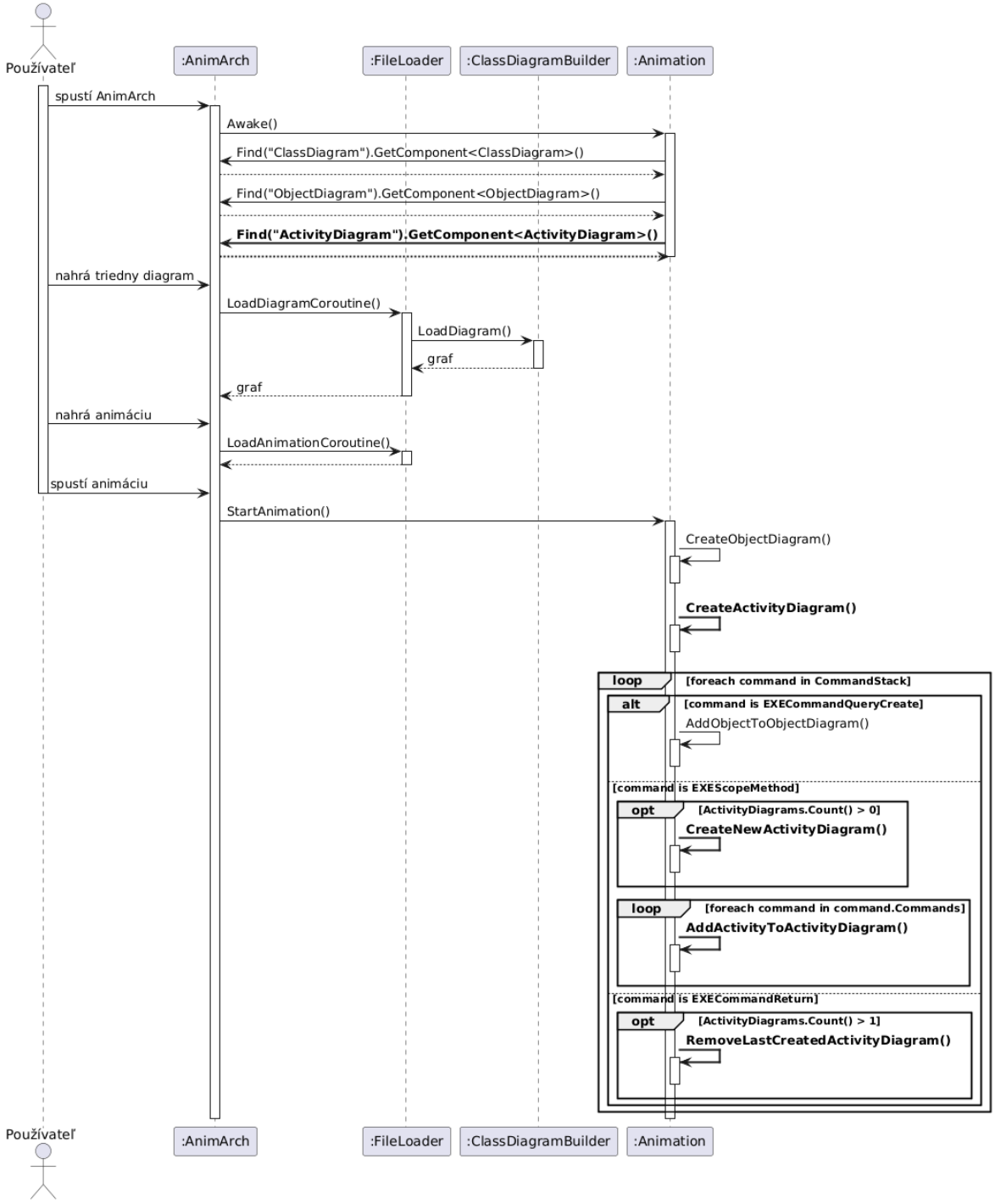
Po spustení animácie sa jednotlivé príkazy z OAL priradia k príslušným triedam reprezentujúcim typ daného príkazu, s ktorými sa v AnimArchu ďalej pracuje. Napríklad, volanie metódy je reprezentované triedou `EXEScopeMethod`, podmienka triedou `EXEScopeCondition`, *foreach* cyklus triedou `EXEScopeForEach` a vrátenie návratovej hodnoty alebo ukončenie metódy je reprezentované triedou `EXECommandReturn`. Ďalej sa vytvorí druhá vrstva pre diagram objektov, kde pridáme taktiež vytvorenie tretej vrstvy pre diagram aktivít. Následne sa postupne vykonávajú všetky príkazy z načítaného kódu.

Počas vykonávania príkazov implementujeme odchytávanie `EXEScopeMethod` príkazov, kedy sa bude vytvárať nový diagram aktivít. Rekurzívne budeme prechádzať cez všetky príkazy danej metódy a každý pridáme do diagramu aktivít. Pred pridaním prvej aktivity taktiež pridáme začiatočný uzol a po poslednej aktivite pridáme koncový uzol. Zároveň tiež budeme pridávať aj šípky medzi aktivitami znázorňujúce riadiaci tok. Ak počas vykonávania príkazov narazíme na ďalší príkaz typu `EXEScopeMethod`, vytvorí sa nový diagram aktivít, ktorý sa bude zobrazovať za pôvodným diagramom aktivít a bude obsahovať príkazy, ktoré sa vykonávajú v tejto metóde.

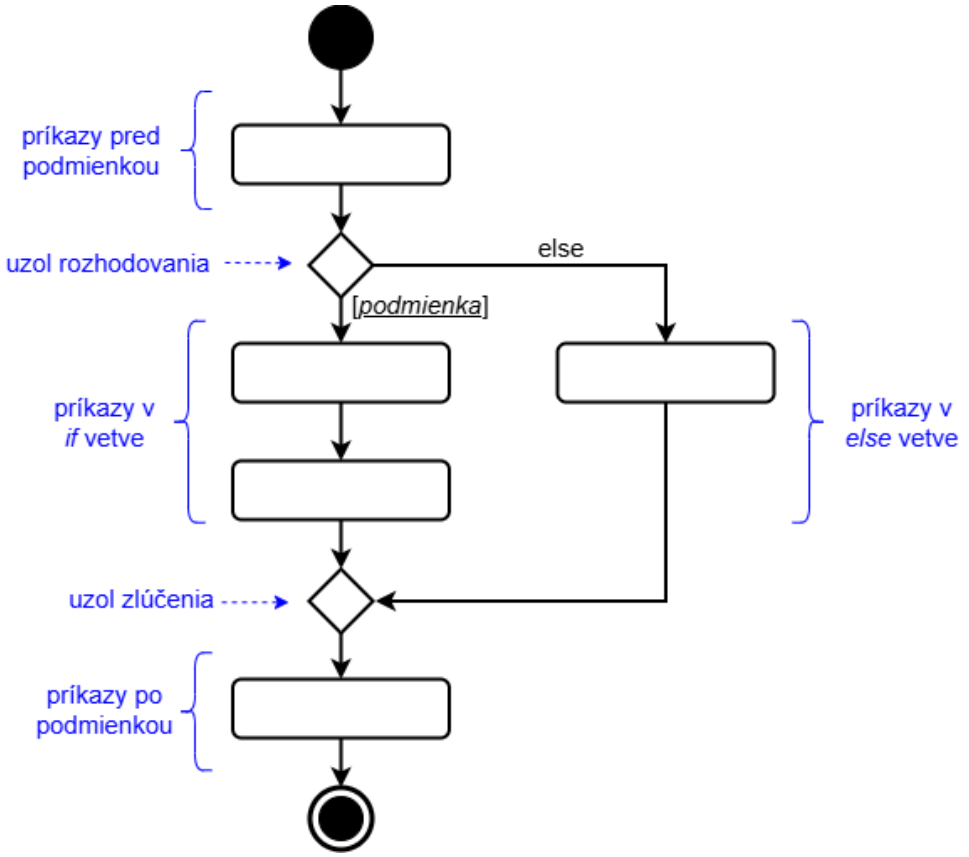
Taktiež budeme zachytávať aj príkazy typu `EXECommandReturn`, ktoré predstavujú koniec vykonávania metódy. Vtedy, ak sa zobrazuje viacero diagramov aktivít, diagram aktivít príslušnej metódy, ktorej patrí daný `EXECommandReturn` príkaz, zmažeme. Ak sa zobrazuje iba jeden diagram aktivít, ten sa mazať nebude, aby ho mal používateľ naďalej k dispozícii aj po skončení vykonávania animácie.

Pri samotnom vytváraní aktivít budeme tiež rozlišovať medzi rôznymi typmi príkazov, pretože vykresľovanie cyklov a podmienok si vyžaduje odlišný prístup v porovnaní s bežnými príkazmi. Bežné príkazy budeme pridávať do diagramu aktivít pod seba.

Pri zobrazovaní podmienky najskôr zobrazíme uzol rozhodovania, ktorý reprezentuje rozvetvenie toku vykonávania podľa splnenia danej podmienky. K uzlu rozhodovania tiež zobrazíme samotnú podmienku v hranatých zátvorkách. Príkazy, ktoré sa vykonajú v prípade, že je podmienka splnená, budú umiestnené pod uzlom rozhodovania. Naopak príkazy, ktoré sa majú vykonať, ak podmienka nie je splnená, budú zobrazené odsadené doprava. Šípku smerujúcu od uzla rozhodovania k príkazu, ktorý sa má vykonať, ak podmienka nie je splnená navyše označíme slovom „*else*“ (v preklade „inak“), pre jasnejšie porozumenie. Po vykreslení všetkých príkazov príslušných k obom vetvám sa tieto vetvy spoja v uzle zlúčenia, pod ktorým budú ďalej umiestnené nasledovné príkazy. Obrázok 3.3 zobrazuje, ako bude vyzerať výsledný diagram aktivít obsahujúci podmienku, ktorý budeme vytvárať.



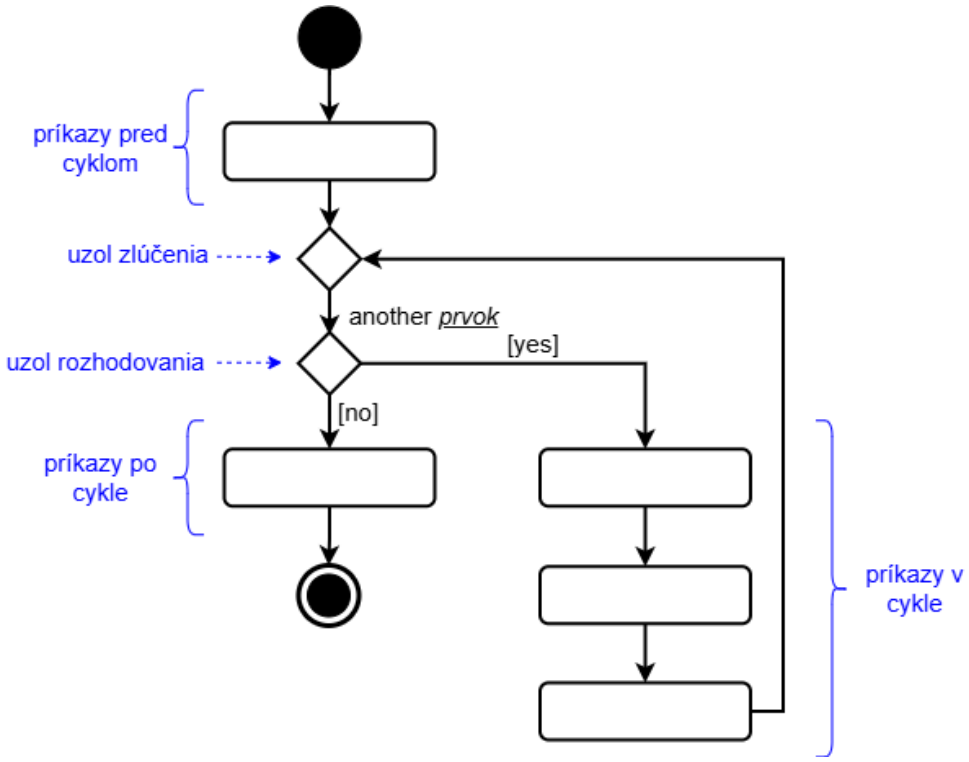
Obr. 3.2: Sekvenčný diagram zobrazujúci proces generovania diagramov v AnimArchu.



Obr. 3.3: Štruktúra diagramu aktivít s podmienkou.

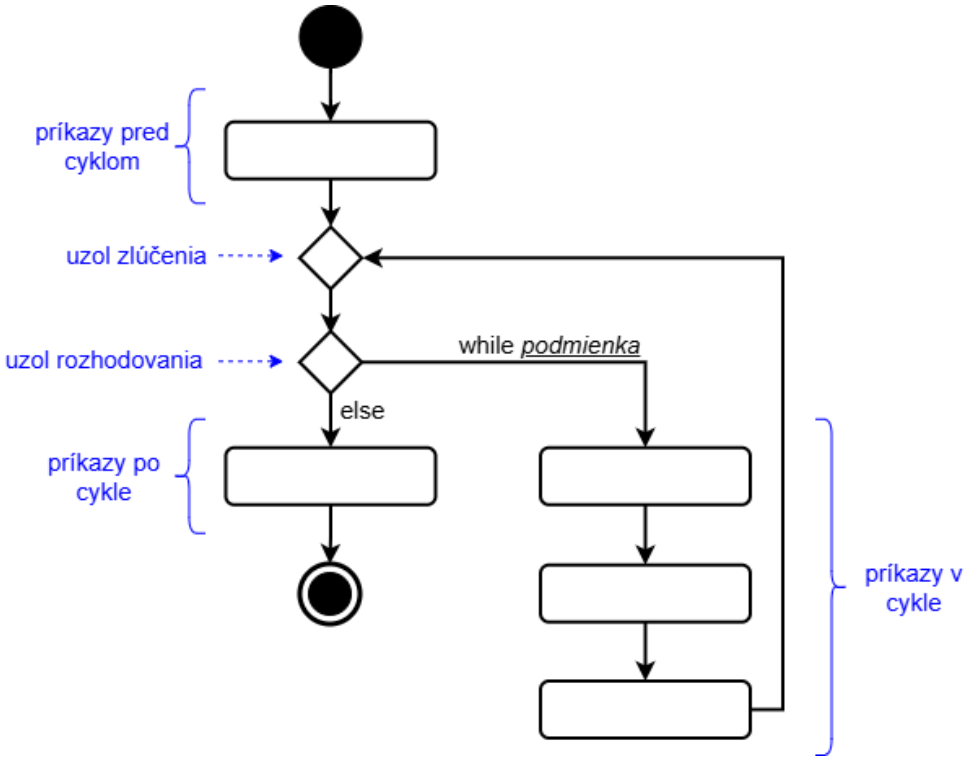
Pri zobrazovaní *foreach* cyklu najskôr zobrazíme uzol zlúčenia, ktorý slúži ako vstupný bod cyklu. Pod tento uzol zobrazíme uzol rozhodovania, v ktorom sa bude tok vykonávania rozvetvovať podľa toho, či existuje ďalší prvok v kolekcii, cez ktorú sa iteruje. Tento uzol rozhodovania označíme textom „*another prvok*“ (v preklade „ďalší *prvok*“), kde za *prvok* dosadíme názov premennej použitej v cykle. Z uzla rozhodovania budú viesť dve šípky. Prvá šípka, označená slovom „*yes*“ (v preklade „áno“), smeruje k aktivitám, ktoré predstavujú telo cyklu. Tieto príkazy zobrazíme odsadené doprava, aby bolo zrejmé, že patria do cyklu. Po vykonaní týchto príkazov sa tok opäť vráti k uzlu zlúčenia, čím sa zabezpečí opakovanie cyklu pre ďalší prvok v kolekcii. Druhá šípka, označená slovom „*no*“ (v preklade „nie“), vedie k aktivitám, ktoré sa vykonajú po skončení cyklu, teda v prípade, že v kolekcii už nie je žiadny ďalší prvok. Tieto príkazy zobrazíme pod rozhodovacím uzlom, čím vizuálne oddelíme časti patriace do cyklu od tých, ktoré nasledujú po jeho ukončení. Výsledný diagram aktivít, ktorý obsahuje *foreach* cyklus je zobrazený na obrázku 3.4.

While cyklus budeme zobrazovať rovnako ako *foreach* cyklus. Hlavný rozdiel bude v označení šípok, kedy šípku vedúcu od uzla rozhodovania k telu cyklu označíme textom „*while podmienka*“ (v preklade „kým *podmienka*“), kde za *podmienka* dosadíme samotnú podmienku *while* cyklu. Šípku vedúcu od uzla rozhodovania k aktivitám, ktoré nasle-



Obr. 3.4: Štruktúra diagramu aktivít s *foreach* cyklom.

dujú po ukončení cyklu, teda v prípade, že podmienka cyklu nie je splnená, označíme slovom „*else*“ (v preklade „inak“). Výslednú štruktúru diagramu aktivít, ktorý obsahuje *while* cyklus vidíme na obrázku 3.5.

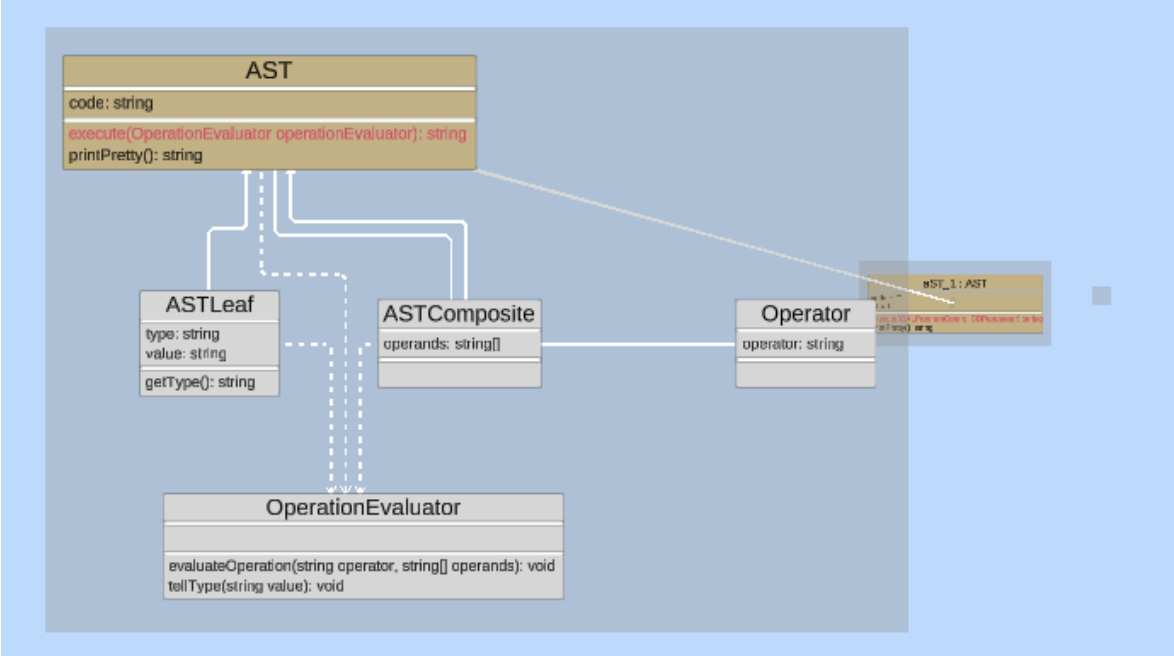


Obr. 3.5: Štruktúra diagramu aktivít s *while* cyklom.

Kapitola 4

Implementácia

Aby sme umožnili vytváranie diagramu aktivít v softvéri AnimArch, najskôr sme pridali tretiu vrstvu pre diagram aktivít, ktorý budeme zobrazovať za diagramom tried a diagramom objektov, ktoré sa už v AnimArchu vizualizujú. Nová vrstva pre diagram aktivít sa vytvorí po spustení animácie, pričom samotné vytváranie diagramu aktivít je popísané v nasledujúcej sekcii 4.1. Rôzne vrstvy pre diagramy tried, objektov a aktivít sú zobrazené na obrázku 4.1.



Obr. 4.1: Ukážka zobrazenia troch vrstiev diagramov. Na prvej vrstve sa zobrazuje diagram tried. Na druhej vrstve sa postupne počas animácie vytvára diagram objektov. Na tretej vrstve sa bude zobrazovať diagram aktivít.

Ďalej sme vytvorili takzvané prefabrikáty, ktoré slúžia ako šablóny pre jednotlivé prvky, ktoré môžeme zobrazovať v diagrame aktivít, konkrétne pre začiatočný a koncový uzol, pre uzly rozhodovania, zlúčenia, paralelného rozvetvenia, paralelného spo-

jenia, a taktiež pre samotné aktivity a šípky označujúce riadiaci tok, pričom sme sa držali zaužívanej notácie, ktorú sme špecifikovali v sekcii 1.5.

4.1 Statické zobrazenie

Diagram aktivít začneme vytvárať po spustení animácie, kedy sa postupne vykonávajú príkazy z načítaného kódu OAL. Pri vykonávaní príkazov sme implementovali odchytávanie príkazov typu `EXEScopeMethod` a `EXECommandReturn`, ako je zobrazené na ukážke pseudokódu 4.1.

V prípade, že práve vykonávaný príkaz je typu `EXEScopeMethod`, teda volanie metódy, vytvoríme nový diagram aktivít, do ktorého zároveň pridáme začiatočný uzol. Ďalej rekurzívne prejdeme cez všetky príkazy metódy a pridáme ich do diagramu aktivít. Budeme tiež rozlišovať medzi rôznymi typmi príkazov, keďže vykreslenie podmienok a cyklov si vyžaduje odlišný prístup. Tento proces rekurzívneho prechádzania a vykresľovania príkazov metódy je zachytený pseudokódom na ukážke 4.2. Po vykreslení poslednej aktivity pridáme aj koncový uzol. Zároveň pridávame aj šípky medzi jednotlivými vrcholmi v diagrame, ktoré znázorňujú riadiaci tok. Ak sa potom opäť vykoná príkaz typu `EXEScopeMethod`, vytvoríme nový diagram aktivít, ktorý zobrazíme v ďalšej vrstve, za pôvodným diagramom aktivít, odkiaľ sa metóda vyvolala. Príkazy v tele vyvolanej metódy ďalej vykresľujeme do novovytvoreného diagramu aktivít.

Keď sa vykonáva príkaz typu `EXECommandReturn`, teda vykonávanie volanej metódy končí, a zobrazujeme viacero diagramov aktivít, príslušný diagram aktivít zmažeme a ďalej budeme zobrazovať už iba pôvodný diagram aktivít. Ak je však zobrazený iba jeden diagram aktivít, ten sa mazať nebude, aby ho mal používateľ naďalej k dispozícii aj po skončení vykonávania.

```
IF CurrentCommand is of type EXEScopeMethod THEN
    Create a new activity diagram
    Add initial activity to diagram
    AddActivity(CurrentCommand)
    Add final activity to diagram
    Add diagram to ActivityDiagramManager stack

ELSE IF CurrentCommand is of type EXECommandReturn THEN
    IF ActivityDiagramManager has more than one diagram THEN
        Remove current diagram from stack
        Set diagram to the peek diagram in stack
    END IF
END IF
```

Alg. 4.1: Pseudokód vytvárania a mazania diagramov aktivít.

```
FUNCTION AddActivity(command)
    IF command is NOT EXEScopeMethod THEN
        SWITCH (type of command)
            CASE EXEScopeCondition:
                AddCondition(command)
                BREAK
            CASE EXEScopeForEach:
                AddForEach(command)
                BREAK
            CASE EXEScopeLoopWhile:
                AddWhile(command)
                BREAK
            DEFAULT:
                Add activity to diagram
                Connect last activity to this one
                Update last activity
                BREAK
        END SWITCH
    ELSE
        FOR EACH subCommand in method
            AddActivity(subCommand)
        END FOR
    END IF
END FUNCTION
```

Alg. 4.2: Pseudokód rekurzívnej metódy, ktorá má na starosti pridávanie aktivít do diagramu.

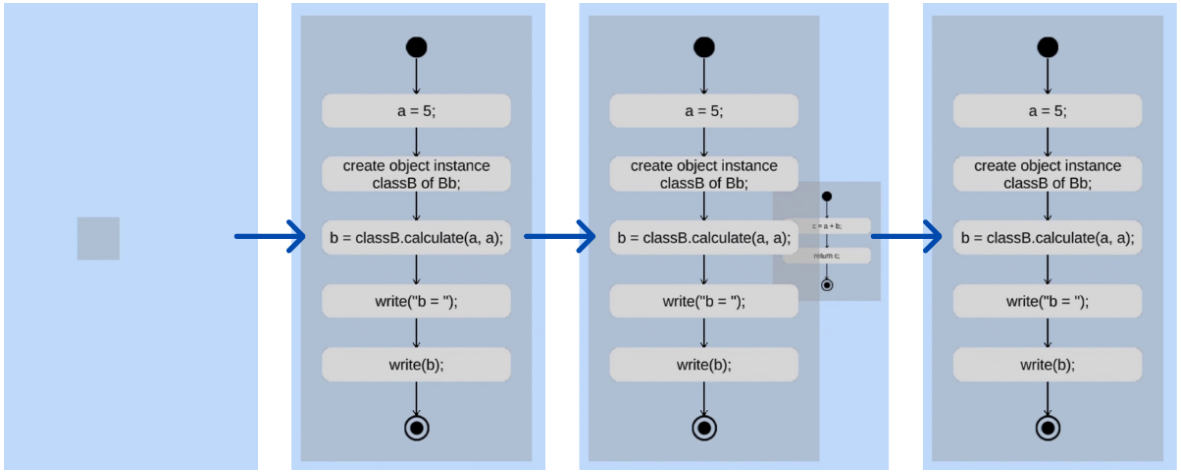
Na obrázku 4.2 môžeme vidieť proces vytvárania a zobrazovania viacerých diagramov aktivít za sebou.

1. V prvom kroku sa po spustení animácie vytvorila prázdna vrstva pre diagram aktivít.

2. V druhom kroku sa počas vykonávania `EXEScopeMethod` príkazu rekurzívne prešlo cez všetky jeho príkazy a vykreslili sa do diagramu aktivít, spolu so začiatočným a koncovým uzlom a šípkami medzi aktivitami.

3. V treťom kroku sa zavolala nová metóda, teda sa opäť vykonal `EXEScopeMethod` príkaz, a teda sa vytvorila nová vrstva a nový diagram aktivít, ktorý opätovne napĺňame rekurzívnym prechádzaním cez príkazy tejto metódy.

4. V štvrtom kroku sa skončilo vykonávanie volanej metódy, teda sa vykonal príkaz typu `EXECommandReturn`. Preto sa príslušný diagram aktivít zmazal a ďalej sa zobrazuje už iba pôvodný diagram aktivít. Ďalej sa pokračuje vo vykonávaní príkazov pôvodnej metódy. Keď sa potom skončí vykonávanie aj tejto metódy, tento diagram aktivít sa mazať nebude a ostane zobrazený aj naďalej.



Obr. 4.2: Proces vytvárania a zobrazovania viacerých diagramov aktivít.

Vykreslenie zložitejších príkazov, ako sú podmienky a cykly, prebieha v samostatných funkciách, pretože si vyžadujú rozličný prístup.

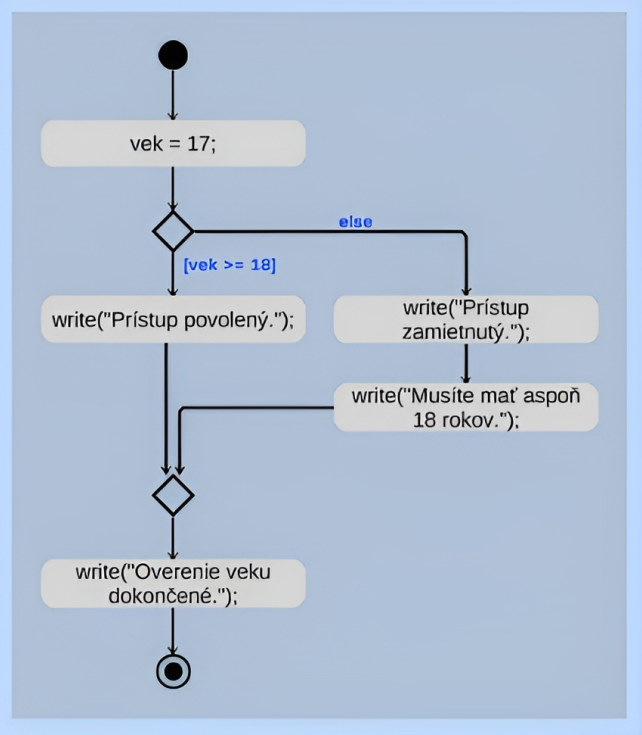
Pri podmienke najskôr vykreslíme uzol rozhodovania, ktorý rozdelí vykonávanie do viacerých vetiev. Príkazy v *if* vetve, teda tie, ktoré sa vykonajú, ak platí podmienka, sa vykreslia pod uzlom rozhodovania. Príkazy, ktoré sa vykonajú, ak podmienka neplatí, sa vykreslia posunuté doprava. Po vykreslení všetkých príkazov v daných vetvách sa tieto vetvy opäť spoja v uzli zlúčenia. Samotnú podmienku zobrazujeme v hranatých zátvorkách pri uzli rozhodovania a šípku smerujúcu od uzla rozhodovania k aktivite, ktorá sa vykoná, ak podmienka nie je splnená, označíme slovom „*else*“ (v preklade „inak“). Pseudokód vykresľovania podmienok je zobrazený na ukážke kódu 4.3.

```
FUNCTION AddCondition(command)
    Add decision node
    Connect last activity to decision node

    FOR EACH subCommand in ifBranch
        AddActivity(subCommand)
    END FOR

    FOR EACH subCommand in elifBranch
        AddActivity(subCommand) // with increased indentation
    END FOR
```

```
    FOR EACH subCommand in elseBranch
        AddActivity(subCommand) // with increased indentation
    END FOR

    Add merge node below all branches
    Connect if, elif and else branches to merge node
    Update last activity
END FUNCTION
```

Alg. 4.3: Pseudokód pridávania podmienky do diagramu aktivít.

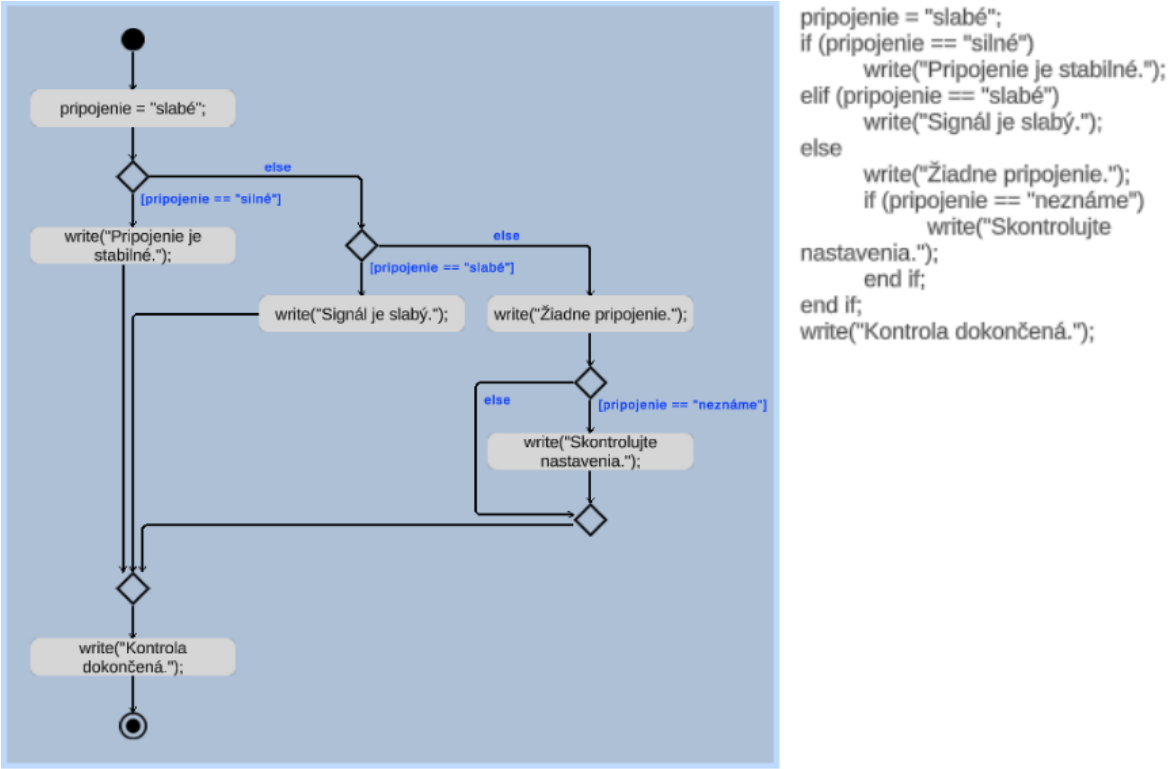
Na obrázku 4.3 je zobrazený vygenerovaný diagram aktivít, ktorý obsahuje jednoduchú podmienku a na obrázku 4.4 vidíme diagram aktivít obsahujúci aj vnorenú podmienku.




```
vek = 17;
if (vek >= 18)
        write("Prístup povolený.");
else
        write("Prístup zamietnutý.");
        write("Musíte mať aspoň 18 rokov.");
end if;
write("Overenie veku dokončené.");
```

Obr. 4.3: Ukážka zobrazenia jednoduchej podmienky v diagrame aktivít.

Pri zobrazovaní *foreach* cyklu najskôr vykreslíme uzol zlúčenia, ktorý predstavuje začiatok cyklu. Následne vykreslíme uzol rozhodovania, kde sa tok rozvetví, podľa toho, či existuje ďalší prvok v kolekcii, cez ktorú sa iteruje. Jednotlivé príkazy v tele cyklu ďalej vykreslíme odsadené doprava. K uzlu rozhodovania tiež pridáme označenie „*another*“ (v preklade „ďalší“) a názov premennej, ktorá sa používa v cykle. Pseudokód zobrazovania *foreach* cyklu je zobrazený na ukážke kódu 4.4. Príklad vytvoreného diagramu aktivít, ktorý obsahuje *foreach* cyklus je zobrazený na obrázku 4.5.

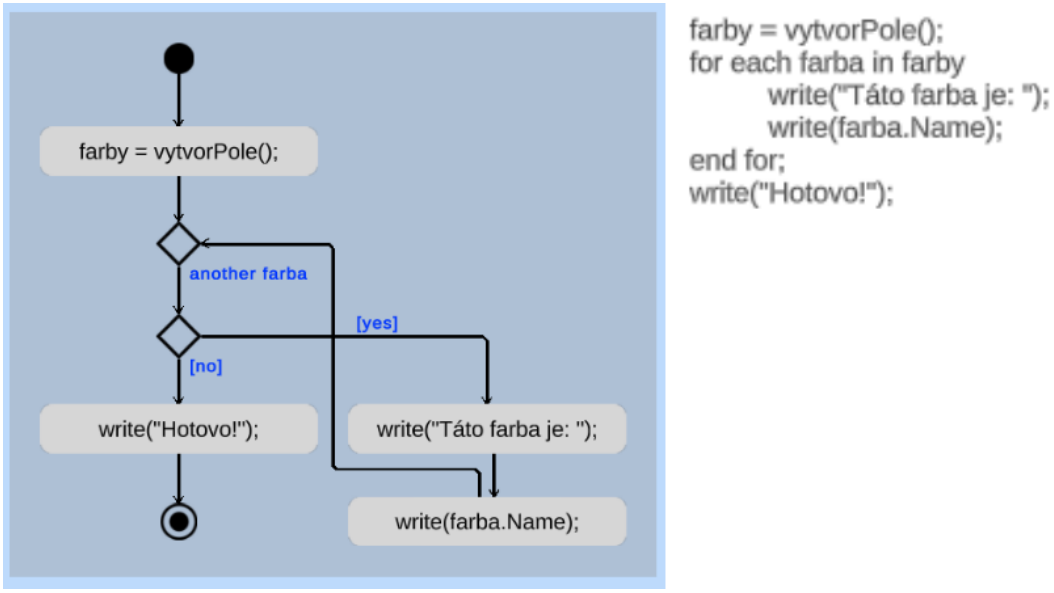


Obr. 4.4: Ukážka zobrazenia vnorenej podmienky v diagrame aktivít.

```
FUNCTION AddForEach(command)
    Add merge node and decision node
    Connect last activity to merge node
    Connect merge node to decision node

    FOR EACH subCommand in loop
        AddActivity(subCommand) // with increased indentation
    END FOR

    Update last activity
END FUNCTION
```

Alg. 4.4: Pseudokód pridávania *foreach* cyklu do diagramu aktivít.

While cykly vytvárame rovnakým spôsobom ako *foreach* cykly, pretože sa líšia len v samotnej logike vykonávania daného kódu. Najskôr teda pridáme uzol zlúčenia, pod ktorý pridáme uzol rozhodovania. Telo cyklu vykreslíme odsadené doprava. Šípku od uzla rozhodovania k telu cyklu označíme textom „*while*“ (v preklade „kým“) a podmienkou cyklu, čím vizuálne odlíšime *while* cyklus od *foreach* cyklu. Šípku od uzla rozhodovania k aktivitám, ktoré sa vykonajú po cykle, teda keď podmienka cyklu nie je splnená, označíme slovom „*else*“ (v preklade „inak“). Na ukážke algoritmu 4.5 je zo-



Obr. 4.5: Ukážka zobrazenia *foreach* cyklu v diagrame aktivít.

brazený pseudokód vytvárania *while* cyklu v diagrame aktivít a na obrázku 4.6 vidíme ukážku vytvoreného diagramu aktivít s *while* cyklom.

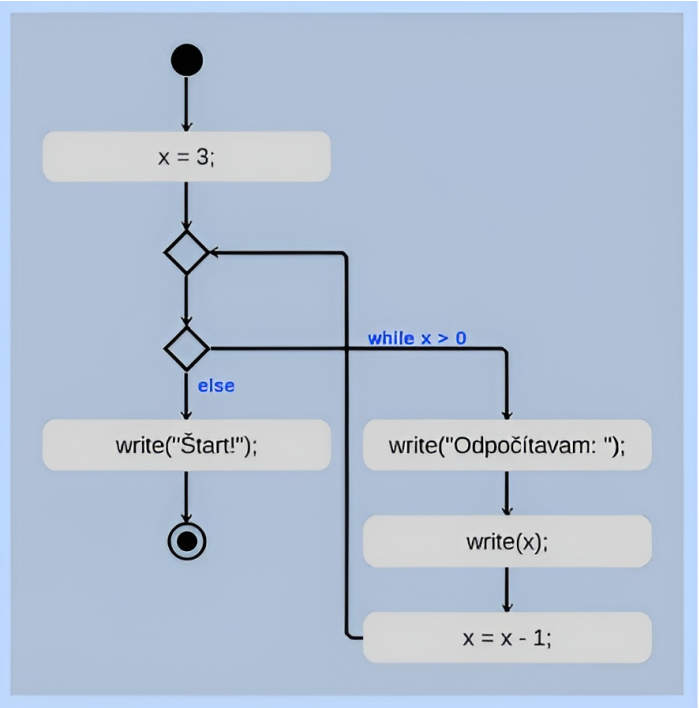
```
FUNCTION AddWhile(command)
    Add merge node and decision node
    Connect last activity to merge node
    Connect merge node to decision node

    FOR EACH subCommand in loop
        AddActivity(subCommand) // with increased indentation
    END FOR

    Update last activity
END FUNCTION
```

Alg. 4.5: Pseudokód pridávania *while* cyklu do diagramu aktivít.

4.2 Dynamické zobrazenie

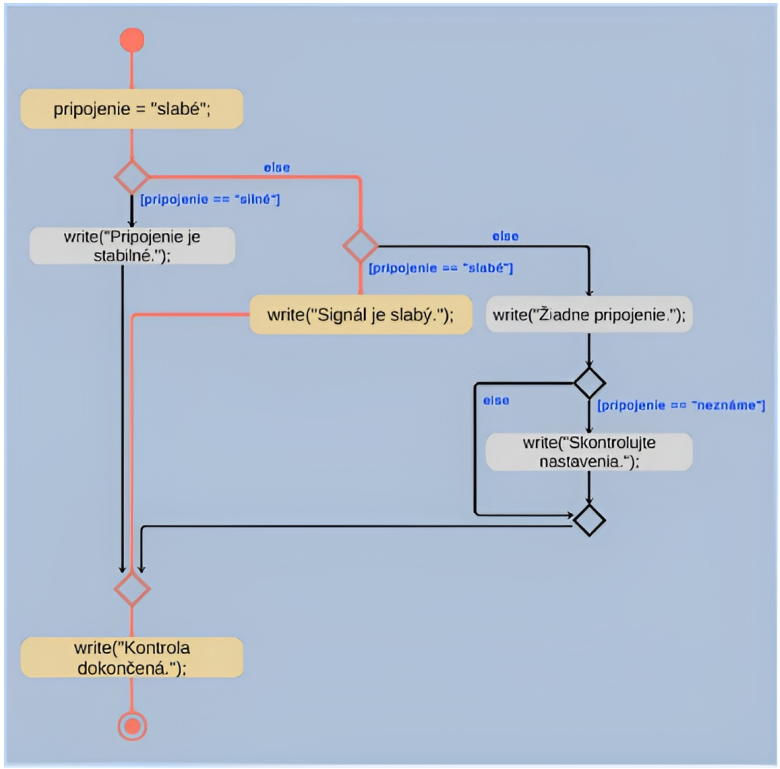
Implementovali sme aj animovanie diagramu aktivít, kedy sa aktivita zodpovedajúca práve vykonávanému príkazu zafarbí, spolu so šípkou smerujúcou k danej aktivite. Pri vykonávaní nejakého príkazu z načítaného kódu sa nájde v diagrame aktivít príslušná aktivita zodpovedajúca danému príkazu a zafarbí sa.

Pri podmienkach jednému príkazu zodpovedajú však dva uzly v diagrame aktivít, a to uzol rozhodovania a uzol zlúčenia. V tomto prípade zafarbíme najskôr iba uzol



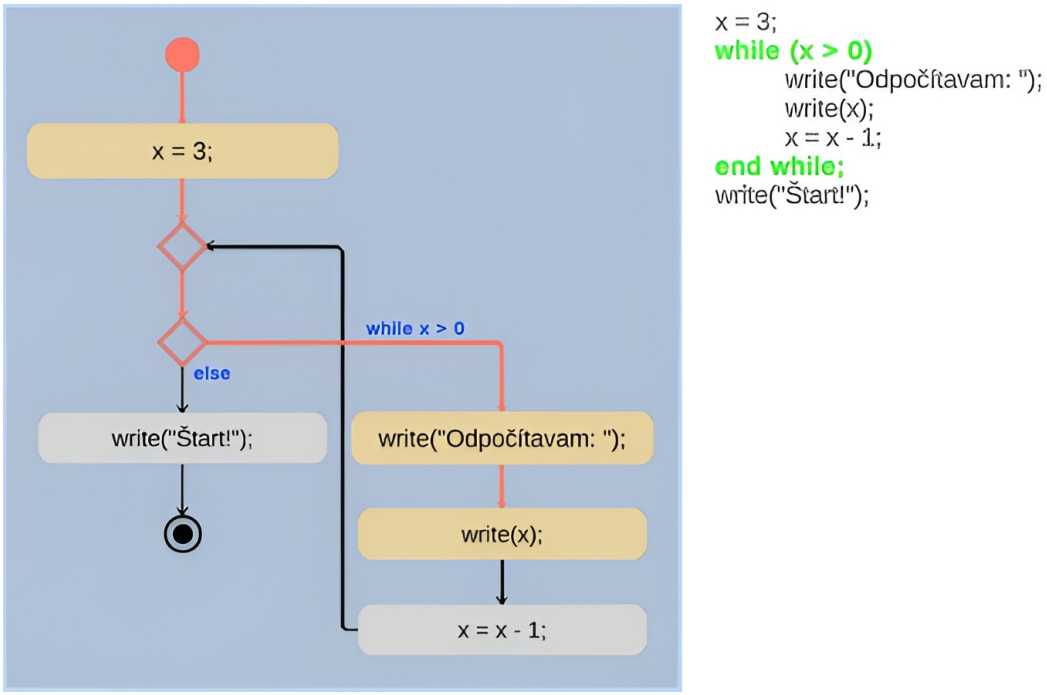
Obr. 4.6: Ukážka zobrazenia *while* cyklu v diagrame aktivít.

rozhodovania. Uzol zlúčenia sa zafarbí, až keď sa vykoná, a teda sa aj zafarbí posledná aktivita v niektorej z vetiev tejto podmienky. Na obrázku 4.7 vidíme, ktorou vetvou podmienky sa vykonával kód, a teda sú zafarbené príslušné aktivity a šípky medzi nimi.



Obr. 4.7: Ukážka animovania diagramu aktivít s podmienkou.

Keďže v cykloch dochádza k opakovanému vykonávaniu príkazov, je potrebné pri každej iterácii najskôr zrušiť zafarbenie všetkých aktivít v tele cyklu a aj šípky medzi nimi. Tým sa zabezpečí, že pri každej iterácii budú tieto prvky zafarbené nanovo, čo bude správne vizualizovať priebeh vykonávania cyklu. Na obrázku 4.8 je zobrazená animácia počas vykonávania príkazov v tele *while* cyklu.



Obr. 4.8: Ukážka animovania diagramu aktivít počas vykonávania tela *while* cyklu.

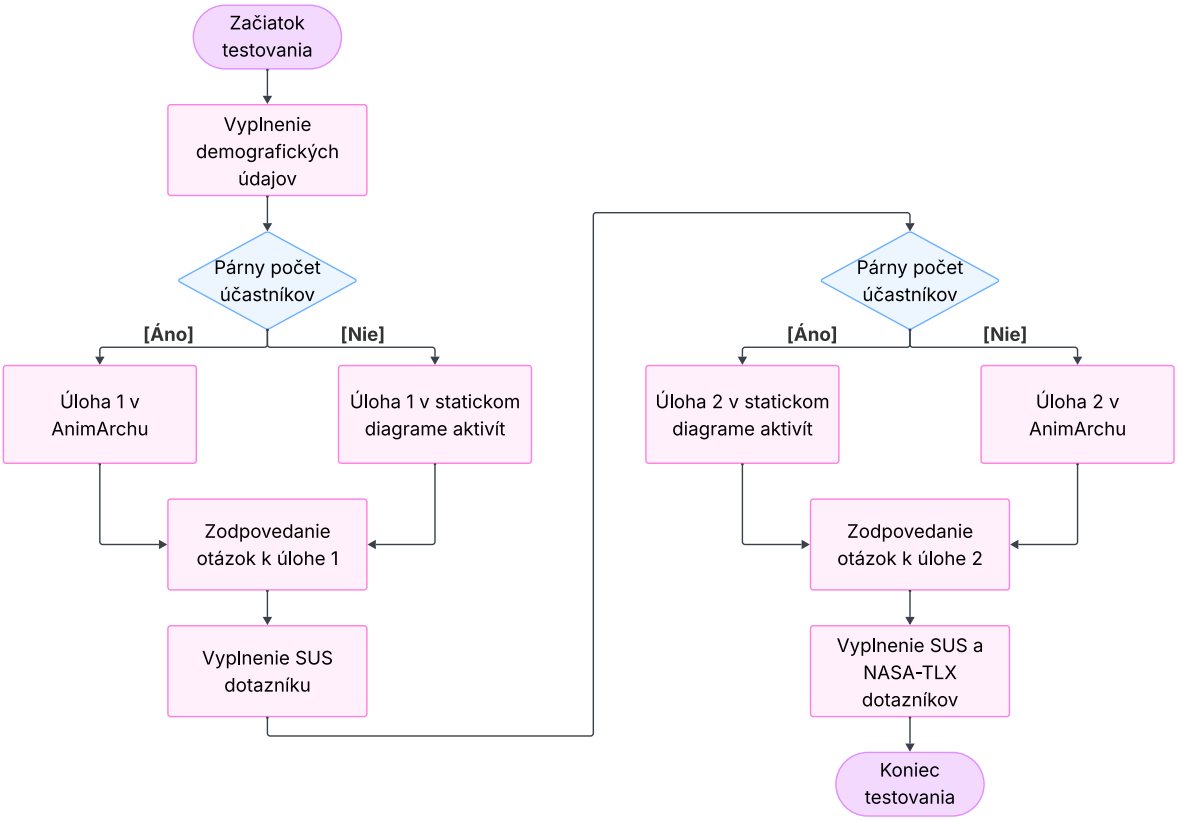
Kapitola 5

Evaluácia výsledkov

V tejto kapitole navrhneme a opíšeme evaluáciu nášho riešenia.

5.1 Návrh testovania

Naše riešenie budeme evaluovať pomocou používateľského testovania, kedy budeme porovnávať úspešnosť riešenia úloh pomocou nášho softvéru v porovnaní so statickým diagramom aktivít. Priebeh testovania je zachytený na obrázku 5.1.



Obr. 5.1: Vývojový diagram opisujúci priebeh testovania.

Účastníci budú po zadaní demografických údajov riešiť dve úlohy, zodpovedajúce

rôznym procesom, ktoré sme vizualizovali pomocou nášho softvéru AnimArch, aj pomocou nástroja, ktorý vygeneroval statický obrázok diagramu aktivít. Každú úlohu budú používatelia riešiť pomocou iného nástroja, teda buď pomocou animovaného diagramu aktivít v AnimArchu alebo pomocou statického diagramu aktivít.

Prvá úloha opisuje proces objednávania tovaru, ktorý zahŕňa rôzne vetvenia toku v prípade, že tovar nie je na sklade alebo platba nie je úspešná. Otázky, ktoré sme pripravili k tejto úlohe sú zobrazené v prílohe A v sekcii Úloha 1. Taktiež sa tam nachádzajú aj vytvorené diagramy aktivít, ktoré budú účastníci využívať. Animovaný diagram aktivít v AnimArchu pre úlohu 1 je zobrazený na obrázku 5.13 a statický diagram aktivít úlohy 1 je zobrazený na obrázku 5.14.

Druhá úloha obsahuje proces vytvorenia zoznamu objednávok, ktoré sa následne spracovávajú v cykle. Otázky k tejto úlohe sú zobrazené v prílohe A v sekcii Úloha 2. Na obrázku 5.15 je zobrazený animovaný diagram aktivít v AnimArchu pre úlohu 2 a statický diagram aktivít je zobrazený na obrázku 5.16.

Používatelia budú mať jednotne stanovený čas, a to minútu a pol, na preštudovanie a pochopenie daného diagramu aktivít. Po tomto čase sa presunú na riešenie úlohy, pozostávajúcej z piatich otázok, ktoré overia, ako dobre si zapamätali a pochopili daný diagram aktivít.

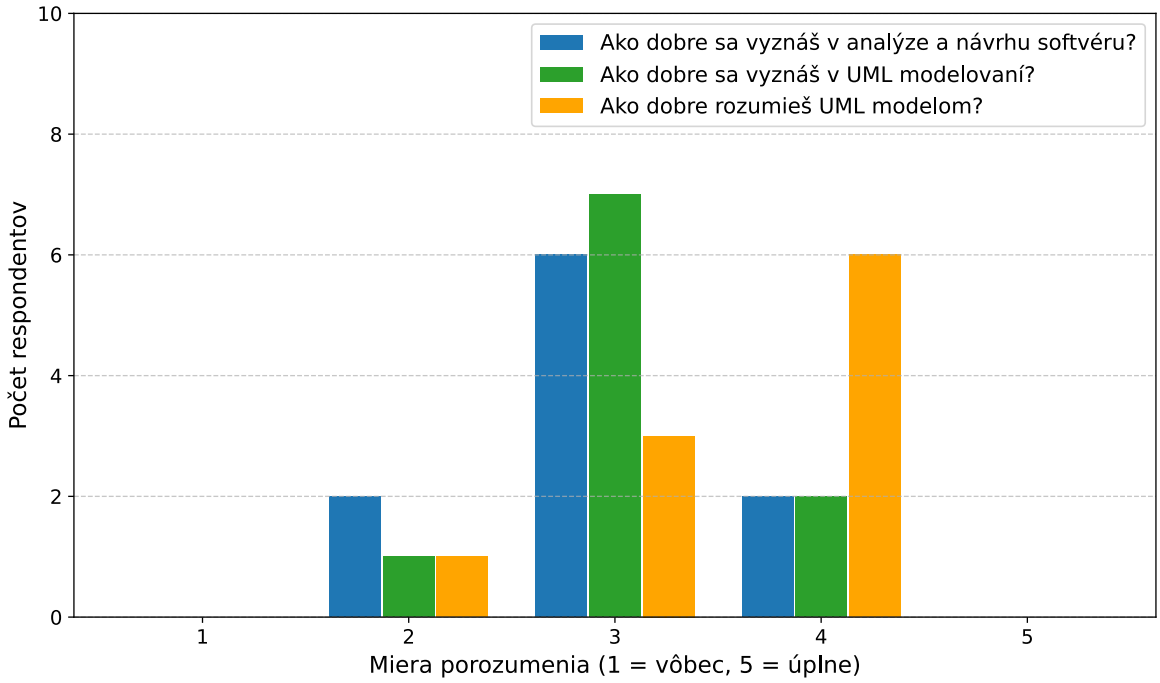
Účastníci budú po každej úlohe odpovedať aj na otázky System Usability Scale (SUS) [29] dotazníku, ktorý bude zisťovať spokojnosť so systémom, ktorý využili. Na záver budú tiež odpovedať na otázky z NASA Task Loader Index (NASA-TLX) [30] dotazníku, pomocou ktorého budeme skúmať, ako záťaž, ktorú účastníci vnímali počas vypracovávania úloh, ktoré riešili pomocou animovaného diagramu v AnimArchu.

5.2 Vyhodnotenie testovania

5.2.1 Demografické údaje účastníkov

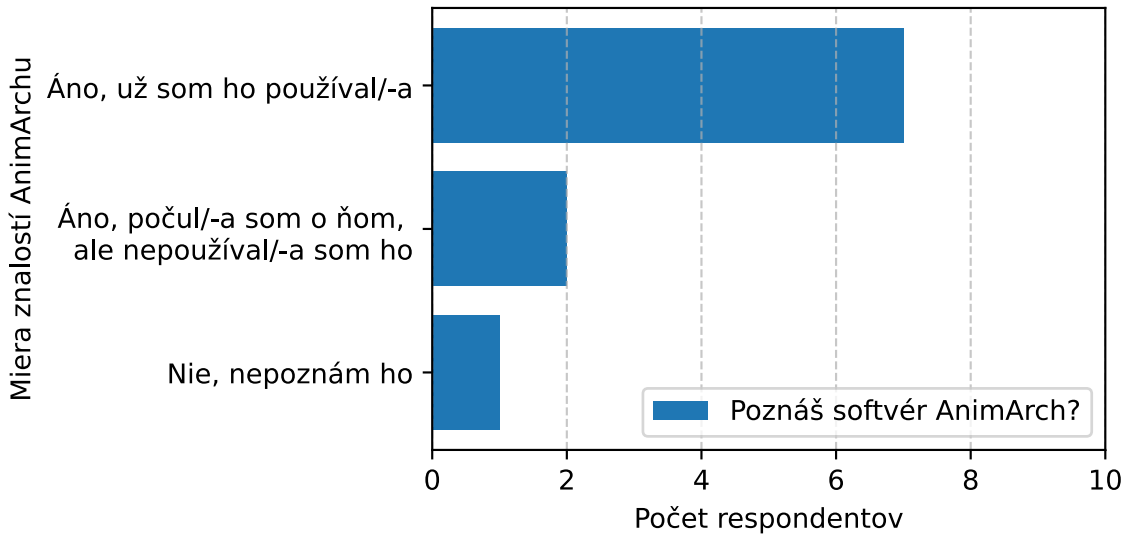
Testovania sa zúčastnilo 10 vysokoškolských študentov aplikovanej informatiky, pričom bolo rovnaké zastúpenie mužov a žien.

Na obrázku 5.2 je zobrazené, ako účastníci ohodnotili svoje znalosti v oblasti analýzy a návrhu softvéru. Nikto neuviedol, že by sa v tejto tematike vyznal úplne, alebo nevyznal vôbec, preto môžeme predpokladať, že úspešnosť používateľov pri riešení úloh nebude ovplyvnená nedostatkom vedomostí z UML modelovania. Odpovede na prvé dve otázky nadobúdajú tvar Gaussovho rozdelenia, čo naznačuje symetrické rozloženie odpovedí okolo priemeru, a teda väčšina z nich považuje svoje znalosti za priemerné. Študenti tiež uviedli, že sa viac vyznajú v UML modelovaní, oproti analýze a návrhu softvéru vo všeobecnosti. Najväčšie znalosti ale uviedli pri samotnom chápaní UML modelov, čo značí, že vedia UML modely skôr chápať ako ich vytvárať.



Obr. 5.2: Subjektívne hodnotenie znalostí účastníkov v analýze a návrhu softvéru.

Účastníkov sme sa tiež pýtali, na ich znalosti softvéru AnimArch. Ako je zobrazené na obrázku 5.3, väčšina z nich uviedla, že AnimArch poznajú a aj ho už používali. Predchádzajúca znalosť AnimArchu ale nemá zásadný vplyv na výsledky testovania, keďže všetci účastníci mali k dispozícii rovnaké inštrukcie a testovací scenár, vďaka čomu boli schopní vykonať úlohy aj bez predchádzajúcich skúseností.

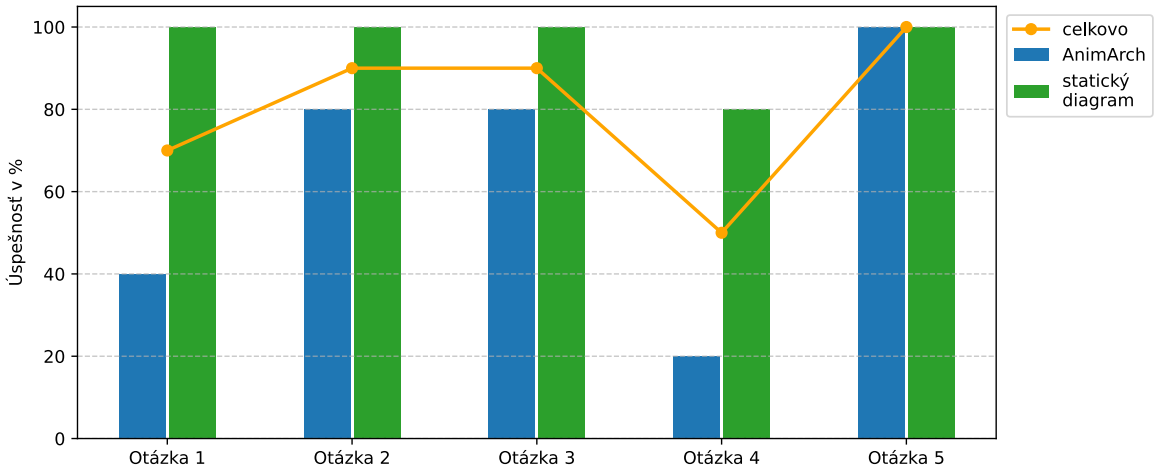


Obr. 5.3: Znalosť softvéru AnimArch.

5.2.2 Vyhodnotenie riešenia úloh

Polovica účastníkov najskôr riešila prvú úlohu pomocou animovaného diagramu aktivít v AnimArchu a následne druhú úlohu pomocou statického diagramu. Druhá polovica účastníkov postupovala opačne, teda prvú úlohu riešila najskôr so statickým diagramom aktivít a druhú úlohu s animovaným diagramom v AnimArchu. Toto rozdelenie umožnilo objektívne porovnať efektivitu oboch typov vizualizácie bez ovplyvnenia postupným poradím riešenia úloh.

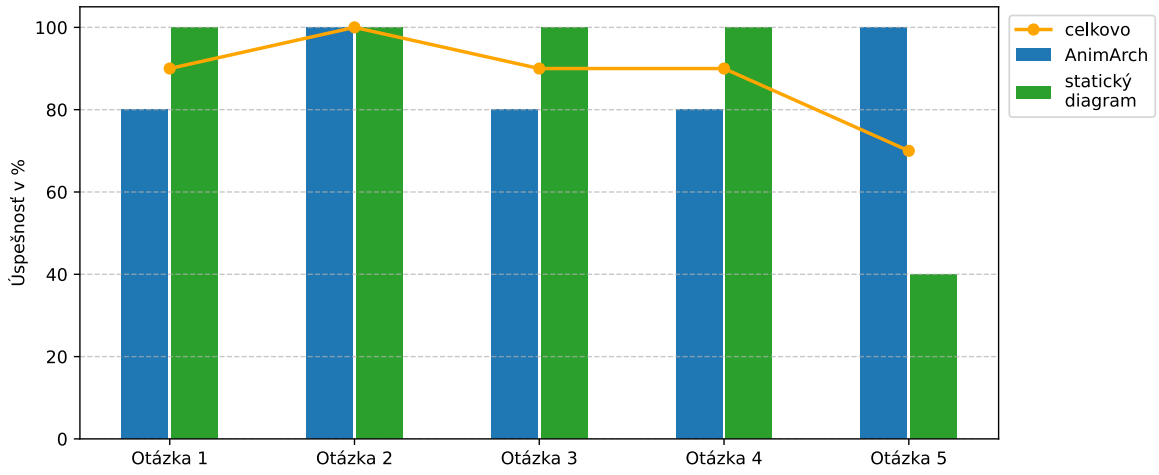
Na obrázku 5.4 je zobrazené porovnanie úspešnosti riešenia otázok v úlohe 1 podľa nástroja, ktorý účastníci využili. Vidíme, že účastníci správne vyriešili viac úloh, ak použili statický diagram aktivít. Môže to byť spôsobené tým, že na tento prístup sú už zvyknutí, a teda sa v ňom orientovali rýchlejšie a s väčšou sebaistotou. Výrazný rozdiel je pri prvej a štvrtej otázke, kedy sa účastníkom darilo lepšie, ak využili statický diagram. Prvá otázka sa zameriavala na akcie, ktoré nastanú, ak tovar nie je skladom. Animácia v diagrame aktivít zobrazovala ale druhú vetvu podmienky, ako môžeme vidieť na obrázku 5.13, je teda možné, že účastníci venovali viac pozornosti práve animovaným aktivitám, a preto nevedeli správne zodpovedať túto otázku. Štvrtá otázka zisťovala, aký krok nastane po úspešnej platbe. V tomto prípade animácia diagramu zobrazovala túto vetvu podmienky, čo vidíme aj na obrázku 5.13. Keďže otázka bola smerovaná na akcie v záverečnej časti diagramu aktivít a účastníci, ktorí odpovedali nesprávne, zvolili možnosť, ktorá sa v diagrame vôbec nevyskytovala, môžeme usúdiť, že používatelia AnimArchu nevenovali dostatočnú pozornosť tejto časti diagramu aktivít, alebo si jej obsah dostatočne nezapamätali.



Obr. 5.4: Úspešnosť riešenia úlohy 1.

Porovnanie úspešnosti riešenia otázok v úlohe 2 je zobrazené na obrázku 5.5. Pri vypracovaní tejto úlohy sa účastníkom v priemere darilo rovnako dobre, bez ohľadu na to, či použili animovaný diagram aktivít v AnimArchu alebo statický diagram. To

môže byť preto, že účastníci po vypracovaní prvej úlohy vedeli, aký štýl otázok môžu očakávať, a teda vedeli, na čo sa majú v diagrame aktivít primárne sústrediť. Výrazný rozdiel v úspešnosti riešenia je pri piatej otázke, ktorá sa zameriavala na akciu, ktorá nastane po spracovaní všetkých objednávok, respektíve po skončení vykonávania cyklu, v ktorom sa objednávky spracúvajú. Vidíme, že 100% účastníkov, ktorí pracovali s animovaným diagramom aktivít, vyriešilo úlohu správne, no iba 40% účastníkov, ktorí používali statický diagram aktivít, bolo úspešných. To mohlo byť spôsobené tým, že zo statického diagramu aktivít nebolo účastníkom zrejmé, kedy nastane koniec cyklu, a teda kedy sa spracujú všetky objednávky. V AnimArchu mohol účastník vďaka animácii jednoduchšie pochopiť, kedy sa spracovali všetky objednávky, a ktorý krok sa následne vykonal. Taktiež v AnimArchu vykresľujeme telo cyklov odsadené doprava, ako môžeme vidieť na obrázku 5.15, čo mohlo byť pre účastníkov viac prehľadné a ľahšie zapamätateľné.



Obr. 5.5: Úspešnosť riešenia úlohy 2.

Môžeme si tiež všimnúť, že používateľom AnimArchu sa pri riešení prvej úlohy darilo menej, ako tím, ktorí využívali AnimArch pri riešení druhej úlohy. To mohlo byť spôsobené tým, že venovali svoju pozornosť aj iným prvkom, ktoré sa v AnimArchu nachádzajú, a teda neboli plne sústredení na samotnú animáciu diagramu aktivít. Pri riešení druhej úlohy už účastníci vedeli, aký štýl otázok môžu očakávať, a teda vedeli na čo sa majú primárne sústrediť, čo mohlo prispieť k vyššej úspešnosti ich odpovedí.

Úspešnosti riešenia otázok pomocou animovaného diagramu aktivít v AnimArchu a pomocou statického diagramu aktivít sme porovnali aj štatistickým testom. Navrhli sme nasledovné hypotézy:

- Nulová hypotéza (H_0): Neexistuje významný rozdiel v úspešnosti úloh pomocou AnimArchu a statického diagramu aktivít.
- Alternatívna hypotéza (H_1): Existuje významný rozdiel v úspešnosti úloh pomo-

cou AnimArchu a statického diagramu aktivít.

Pre každý nástroj sme vytvorili vektor reprezentujúci skóre účastníkov, ktorí používali daný nástroj. Skóre pre každého účastníka sa vypočíta ako počet správnych odpovedí pri riešení jednej úlohy. Výsledné vektory sú:

$$animarchSkore = [4, 5, 4, 3, 3, 5, 3, 4, 2, 5]$$
$$diagramSkore = [4, 5, 4, 4, 4, 5, 5, 5, 5, 5].$$

Na overenie nulovej hypotézy sme najskôr vykonali Shapiro-Wilkov test normality s hodnotou $\alpha = 0.05$, kedy nám vyšli hodnoty $p = 0.19099$ a $p = 0.00017$. Keďže jedna hodnota p je menšia ako 0.05 dáta nie sú normálne rozdelené. Preto použijeme Wilcoxonov test, ktorý je neparametrický a vhodný pre porovnanie dvoch skupín, práve ak dáta nie sú normálne rozdelené. Po použití Wilcoxonovho testu nám vyšla hodnota $p = 0.0625$, čo je väčšie ako hranica α, a preto nulovú hypotézu nezamietame. Dospejeme teda k záveru, že neexistuje štatisticky významný rozdiel v úspešnosti riešenia úloh pomocou rôznych diagramov aktivít.

Keďže nulová hypotéza nebola zamietnutá, na dokázanie alternatívnej hypotézy nemáme dostatok údajov.

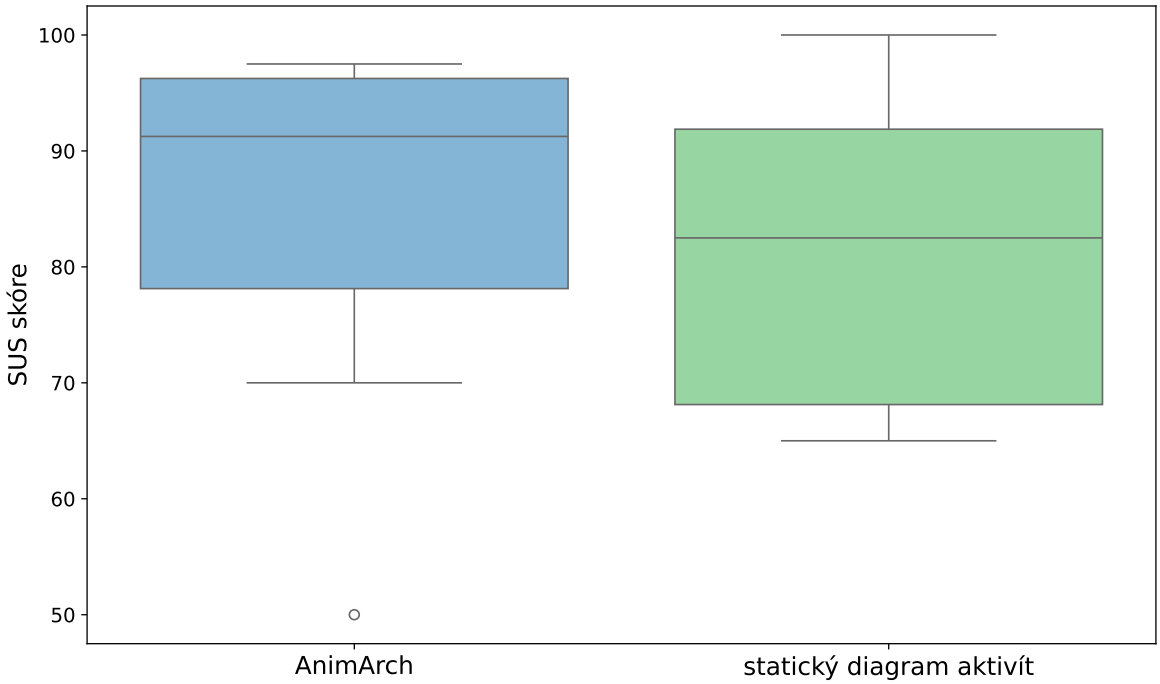
Aj napriek tomu, že používatelia statického diagramu aktivít boli úspešnejší pri vyššom počte otázok boli, neznamená to, že animovaný diagram aktivít je neefektívny. Naopak, keďže ide o nový prístup, na ktorý používatelia nie sú zvyknutí, vyžaduje si určitý čas na osvojenie. Po oboznámení sa s takýmto interaktívnym spôsobom vizualizácie diagramu aktivít by jeho potenciál mohol výraznejšie vyniknúť, najmä pri komplexnejších procesoch, kedy by animácia výrazne zjednodušila pochopenie daného procesu.

5.2.3 Vyhodnotenie SUS dotazníka

SUS je nástroj na meranie použiteľnosti systému, ktorý pozostáva z desiatich tvrdení hodnotených na päťstupňovej Likertovej škále. Používatelia pri odpovediach vyberajú hodnoty od 1 do 5, kde 1 znamená „Úplne nesúhlasím“ a 5 „Úplne súhlasím“. Výsledkom je číselné skóre od 0 do 100, ktoré odráža mieru použiteľnosti systému z pohľadu používateľa. Za priemerné SUS skóre sa pokladá 68 [29], pričom ak je skóre v rozmedzí od 50 do 70, použiteľnosť sa považuje za priemernú alebo akceptovateľnú. SUS skóre pod hranicou 50 naznačuje slabú mieru použiteľnosti systému. Naopak SUS skóre v rozmedzí 70 až 80 indikuje dobrú použiteľnosť systému a skóre nad 80 značí výbornú úroveň použiteľnosti.

SUS skóre vypočítame na základe pozícií odpovedí účastníkov na škále. Pri párnych otázkach získame skóre odpočítaním danej hodnoty odpovede od 5. Pri nepárnych

otázkach odrátame 1 od hodnoty odpovede. Výsledné skóre získame sčítaním týchto hodnôt a vynásobením číslom 2.5, čím dosiahneme, že výsledné skóre bude v rozmedzí 0 až 100.



Obr. 5.6: Krabicový diagram SUS skóre pre AnimArch a statický diagram aktivít.

Na obrázku 5.6 je porovnanie vypočítaných SUS skóre pre softvér AnimArch a pre statický diagram aktivít.

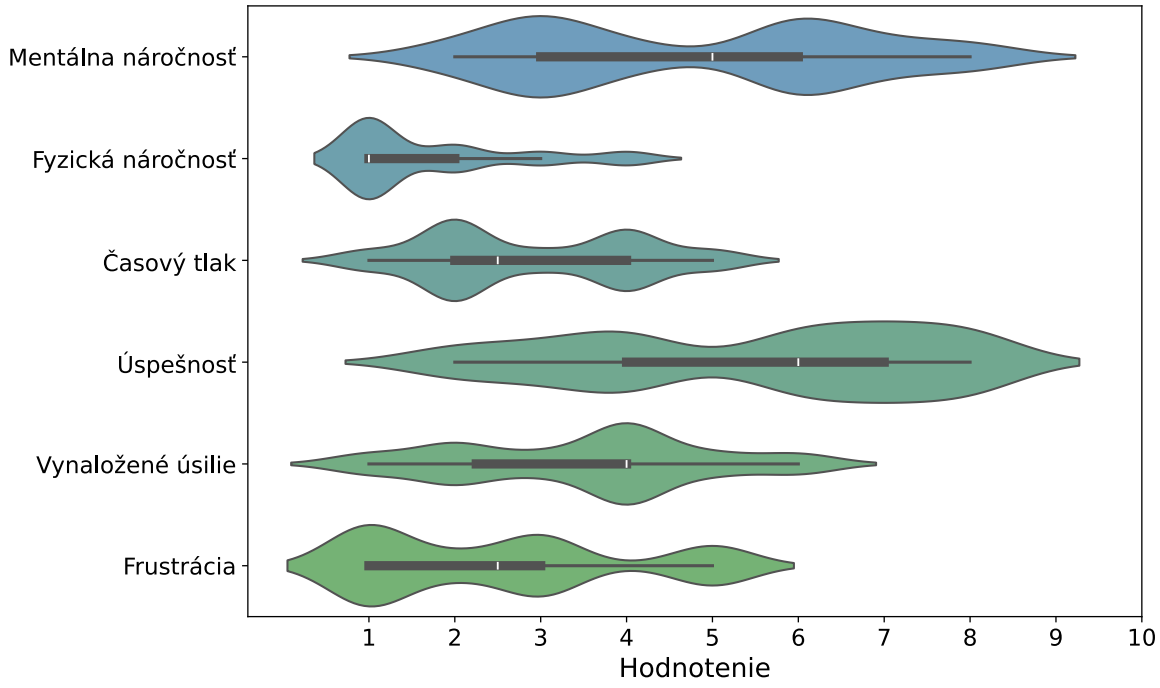
Skóre pre AnimArch sa pohybovali v rozmedzí 50 až 97.5, pričom skóre 50 predstavuje odľahlú hodnotu (outlier), ktorá sa výrazne líši od ostatných hodnôt. Medián SUS skóre pre AnimArch bol 91.25, čo sa považuje za výbornú použiteľnosť.

SUS skóre pre statický diagram aktivít malo hodnoty 65 až 100, pričom medián bol 82.5, čo sa tiež pokladá za výbornú mieru použiteľnosti.

Aj napriek tomu, že používatelia pracovali s animovaným diagramom aktivít v AnimArchu prvýkrát, použiteľnosť hodnotili kladne, čo môže byť náznak toho, že sme naše riešenie navrhli a implementovali vhodne a užívateľsky prívetivo.

5.2.4 Vyhodnotenie NASA-TLX dotazníka

NASA-TLX je nástroj, ktorý sa využíva pri meraní záťaže, ktorú človek vníma pri vykonávaní nejakej úlohy. Obsahuje 6 otázok, ktoré zisťujú do akej miery človek pociťoval mentálnu a fyzickú náročnosť, časový tlak, ako hodnotí svoju úspešnosť, vynaložené úsilie a frustráciu počas vykonávania danej úlohy. Používatelia hodnotili tieto aspekty vzhľadom na úlohu, ktorú riešili pomocou softvéru AnimArch, pričom využívali 10-bodovú škálu.



Obr. 5.7: Husľový diagram NASA-TLX skóre pre AnimArch.

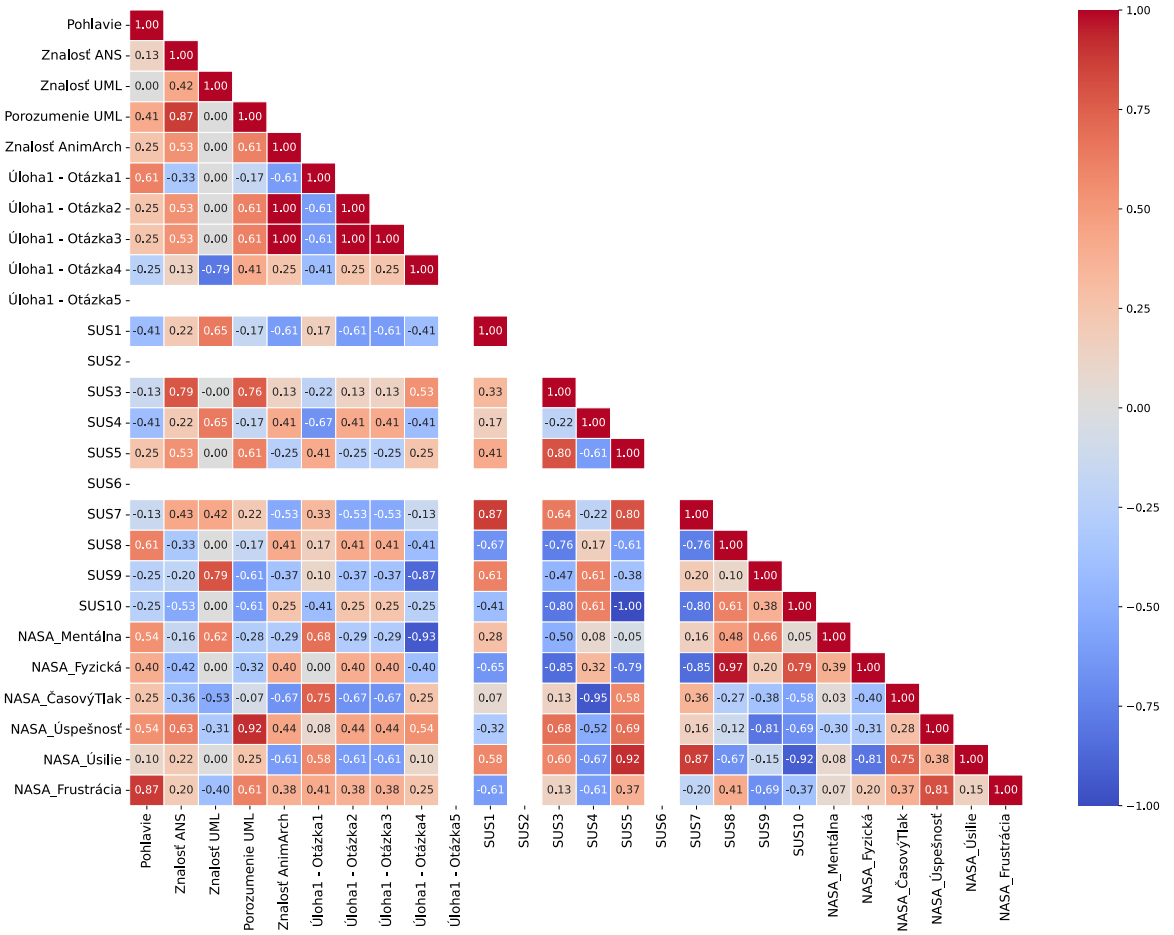
Na obrázku 5.7 je zobrazené hodnotenie jednotlivých aspektov, pričom väčšia hodnota znamená horšie hodnotenie, teda napríklad väčšiu náročnosť, časový nátlak alebo väčší pocit frustrácie. Niektorí používatelia uviedli nižšiu úroveň mentálnej náročnosti, zatiaľ čo iní pociťovali vyššiu náročnosť. Tento rozdiel môže súvisieť s ich predchádzajúcimi znalosťami UML modelovania. Používatelia, ktorí svoje znalosti ohodnotili ako podpriemerné, mohli mať väčší problém s porozumením diagramu aktivít a následným riešením úloh. Naopak, tí, ktorí sa považovali za skúsenejších, mohli vnímať úlohy ako menej náročné. Používatelia prevažne nepociťovali fyzickú náročnosť ani frustráciu pri riešení úlohy, čo môže znamenať, že s naším softvérom sa im pracovalo dobre a bez výraznejších problémov. Používatelia svoju úspešnosť hodnotili rôzne, čo tiež môže súvisieť s hodnotením mentálnej náročnosti, kedy mohli používatelia, ktorým prišla úloha náročnejšia, označiť svoju úspešnosť za horšiu. Hodnotenie úspešnosti má však v dotazníku opačné bodovanie, na aké môžu byť používatelia bežne zvyknutí, keďže vyššia hodnota v tomto prípade predstavuje horšiu úspešnosť. Je teda možné, že niektorí účastníci si túto skutočnosť neuvedomili, a preto uviedli vyššie číslo, aj keď mysleli jeho opačnú hodnotu.

5.2.5 Celkové vyhodnotenie

Na základe výsledkov testovania sme vytvorili viacero korelačných teplotných máp, na ktorých znázorňujeme vzťahy medzi rôznymi dátami. Pri vytváraní týchto máp sme využili Pearsonov korelačný koeficient, ktorý meria lineárnu závislosť medzi dvoma

premennými. Hodnota tohto koeficientu nadobúda hodnoty v rozmedzí od -1 do 1, kde hodnota 1 predstavuje pozitívnu koreláciu, čo znamená, že skúmané premenné sú navzájom priamo úmerné. Hodnota -1 predstavuje negatívnu koreláciu a znamená, že premenné sú nepriamo úmerné. Hodnota 0 značí, že medzi premennými nie je žiadna lineárna závislosť.

Pri značení jednotlivých dát v mapách sme, pre lepšiu prehľadnosť, využili skrátené označenia. Označenie *Znalosť ANS* predstavuje ako účastníci ohodnotili svoju znalosť analýzy a návrhu softvéru. Označenia *Znalosť UML* a *Porozumenie UML* označujú ako účastníci ohodnotili svoju znalosť a porozumenie UML modelov. Označenia *SUS1* až *SUS10* predstavujú štandardné otázky, ktoré sa využívajú v SUS dotazníkoch. Otázky z NASA-TLX sú na obrázku skrátene označené, podľa aspektov, na ktoré sa zameriavajú, pričom sú bližšie opísané v sekcii 5.2.4. Riadky a stĺpce, pre ktoré sa korelácia nedala určiť z dôvodu konštantných hodnôt, sa v korelačnej mape zobrazujú prázdne.



Obr. 5.8: Korelačná teplotná mapa znázorňujúca vzťahy medzi úspešnosťou riešenia úlohy 1 pomocou AnimArchu a zvyšnými dátami z dotazníku.

Na obrázku 5.8 je znázornená korelačná teplotná mapa, ktorá zachytáva vzťahy medzi úspešnosťou riešenia úlohy 1, keď účastníci využívali animovaný diagram ak-

tivít v AnimArchu, a demografickými údajmi, či odpoveďami na SUS a NASA-TLX dotazníky.

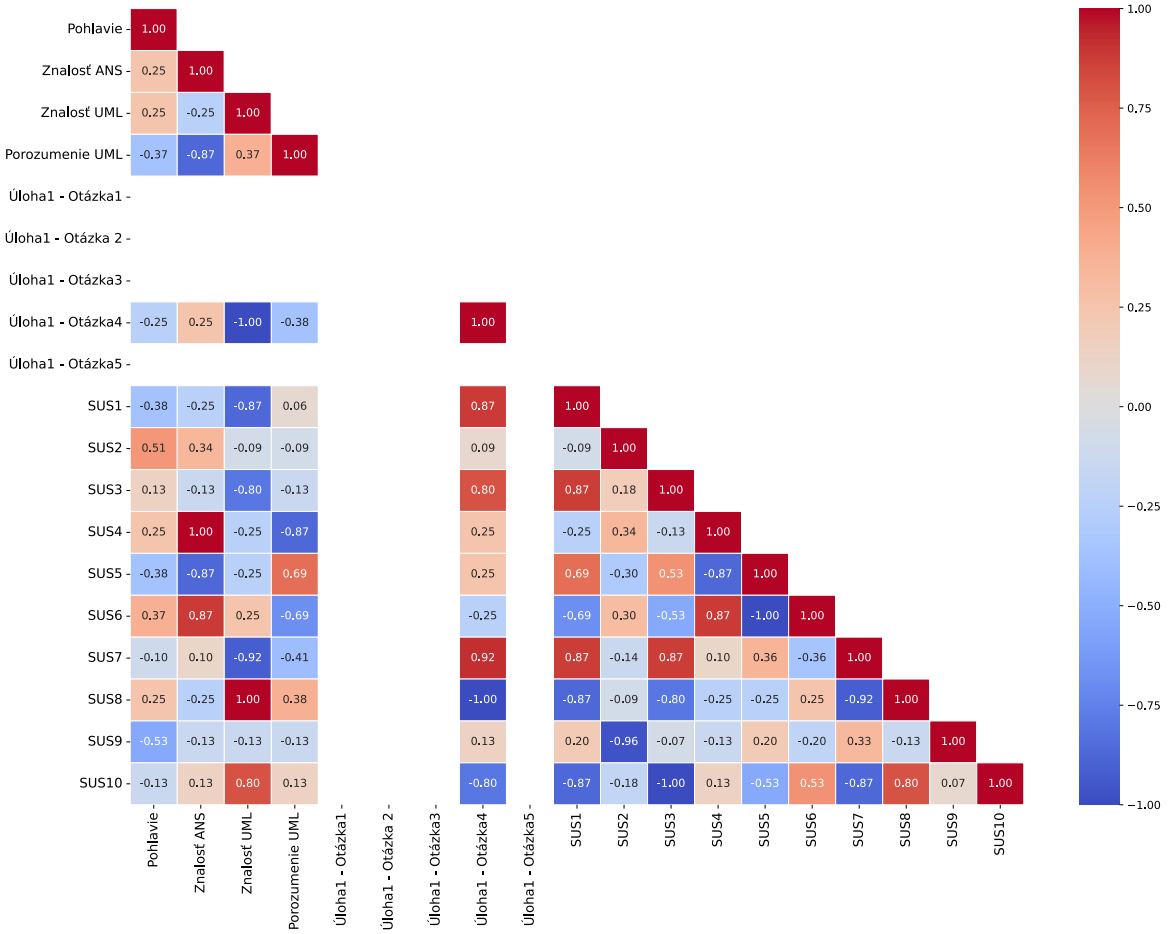
Korelácia medzi znalosťou analýzy a návrhu softvéru a porozumením UML zobrazuje, že čím väčšiu majú účastníci znalosť analýzy a návrhu, tým lepšie vedia chápať UML modely.

Otázky 2 a 3 úlohy 1 overovali, či účastníci správne pochopili podmienky v diagrame aktivít. Vidíme, že práve úspešnosť riešení týchto otázok významne koreluje so znalosťami AnimArchu, čo naznačuje, že používateľom sa darilo lepšie, ak mali predošlé skúsenosti s AnimArchom. To môže byť spôsobené tým, že sa v softvéri cítili pohodlnejšie, keďže ho viac poznali. Znalosti analýzy a návrhu softvéru, či UML modelov nemali príliš významnú koreláciu s úspešnosťou riešenia úlohy 1.

Medzi úspešnosťou riešenia otázky 4 a otázkou 9 zo SUS dotazníka je negatívna korelácia, ktorá predstavuje, že používatelia boli menej úspešní, hoci uviedli, že sa pri používaní systému cítili sebavedomo. To môže naznačovať, že systém sa im používal dobre, ale na správne vyriešenie otázok si diagram aktivít dostatočne nezapamätali alebo ho nepochopili. Negatívna korelácia medzi otázkami 5 a 10 zo SUS dotazníka naznačuje, že čím menej sa zdal účastníkom systém dobre prepojený, tým viac vecí sa museli naučiť pred použitím systému. To môže naznačovať, že systém bol pre niektorých používateľov komplexný a menej prehľadný. Keďže ale animovanie diagramu aktivít v AnimArchu bolo pre účastníkov novou skúsenosťou, je možné, že po dlhšom čase by si účastníci systém osvojili a mohli by ho plnohodnotnejšie využívať.

Mentálna náročnosť úlohy negatívne koreluje s úspešnosťou riešenia otázky 4, čo značí, že účastníci, ktorí boli pri riešení otázky 4 menej úspešní pokladali úlohu za náročnejšiu. Korelácia medzi fyzickou náročnosťou a SUS otázkou 8 predstavuje, že používatelia hodnotili riešenie úlohy ako málo fyzicky náročné, a zároveň ohodnotili systém ako ľahko ovládateľný. Z toho môžeme usúdiť, že animovaný diagram aktivít sme navrhli vhodne, tak aby sa používateľom ľahko používal. Negatívna korelácia medzi pocitom časového nátlaku a SUS otázkou 4 naznačuje, že účastníci, ktorí pociťovali vyšší časový tlak, uviedli, že nemali pocit potreby pomoci technicky zdatnejšej osoby. Z toho môžeme usúdiť, že hoci účastníci nepotrebovali pomoc technickejšej osoby, potrebovali by viac času na osvojenie si systému. Korelácia ohodnotenia svojej úspešnosti a porozumenia UML modelov značí, že účastníci, ktorí označili svoje znalosti chápania UML modelov za vyššie, hodnotili svoju úspešnosť horšie. Avšak, ako sme opísali v predchádzajúcej sekcii, je možné, že používatelia nesprávne pochopili hodnotenie úspešnosti, a preto nevieme s určitosťou analyzovať túto vlastnosť. Korelácia medzi vynaloženým úsilím a SUS otázkou 5 zobrazuje, že používatelia, ktorí vynaložili viac úsilia pri riešení úlohy, mali väčší pocit, že systém je dobre prepojený. Hodnotenie vynaloženého úsilia tiež negatívne koreluje s otázkou 10 zo SUS dotazníka, kedy používatelia, ktorí vynaložili viac úsilia sa museli naučiť menej vecí, pred riešením úlohy.

To môže súvisieť s tým, že disponovali dostatočnými znalosťami alebo si systém viac osvojili.



Obr. 5.9: Korelačná teplotná mapa znázorňujúca vzťahy medzi úspešnosťou riešenia úlohy 1 pomocou statického diagramu a zvyšnými dátami z dotazníku.

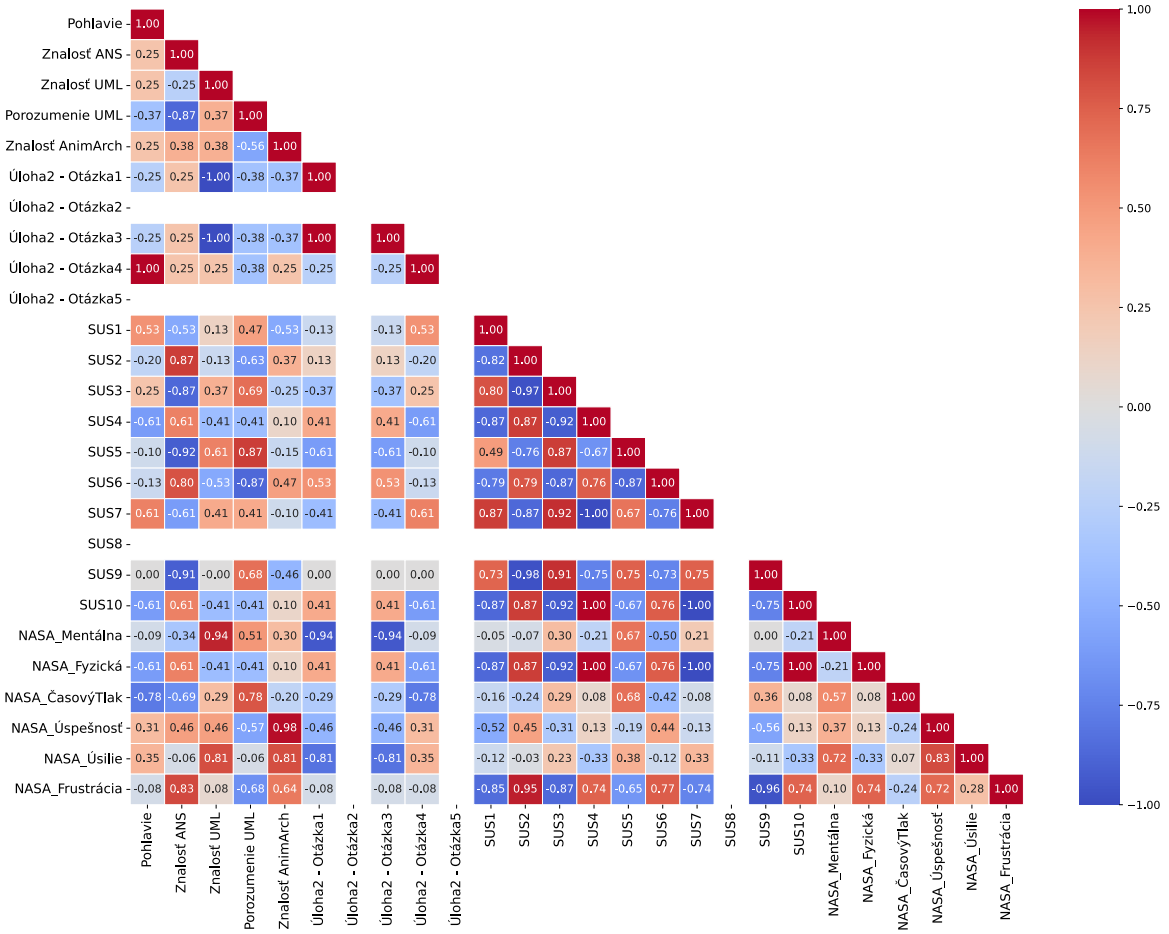
Na obrázku 5.9 je znázornená korelačná teplotná mapa, zachytávajúca vzťahy medzi úspešnosťou riešenia úlohy 1, keď účastníci využívali statický diagram aktivít, a demografickými údajmi a odpoveďami na SUS dotazník.

Výrazná negatívna korelácia je medzi úspešnosťou riešenia otázky 4 a znalosťou UML modelov, kedy účastníci pravdepodobne precenili svoje znalosti a darilo sa im horšie, keď uviedli vyššie znalosti UML modelov.

Otázka 4 zo SUS dotazníku výrazne koreluje s uvedenými znalosťami analýzy a návrhu softvéru, čo naznačuje priamu úmeru medzi znalosťami účastníka a pocitom potreby technicky zdatnejšej osoby pri používaní statického diagramu, aj keď by sme predpokladali opak. Znalosť UML modelovania tiež významne koreluje s odpoveďou 8 na SUS dotazník, kedy čím vyššiu znalosť účastníci uviedli, tým ťažšie sa im statický diagram ovládal. Otázka 8 tiež výrazne negatívne koreluje s úspešnosťou riešenia otázky 4, ktorá zisťovala, či účastníci správne pochopili vykreslenie podmienky v dia-

grame. To značí, že používatelia, ktorým sa ťažšie ovládal statický diagram boli aj menej úspešní pri riešení otázky 4. Z toho môžeme usúdiť, že statický diagram aktivít nebol dostačujúci, na to aby ho účastníci správne interpretovali, a teda boli pri riešení otázky úspešní. Taktiež negatívne korelujú SUS otázky 3 a 10, kedy účastníci, ktorým sa systém ľahko používal, mali aj pocit, že sa museli naučiť menej vecí, aby ho mohli používať.

Korelačná teplotná mapa zobrazujúca vzťahy medzi úspešnosťou riešenia úlohy 2, ktorú účastníci riešili pomocou animovaného diagramu aktivít, a demografickými údajmi a odpoveďami na SUS a NASA-TLX dotazníky je zobrazená na obrázku 5.10.



Obr. 5.10: Korelačná teplotná mapa znázorňujúca vzťahy medzi úspešnosťou riešenia úlohy 2 pomocou AnimArchu a zvyšnými dátami z dotazníku.

Zaujímavá je negatívna korelácia medzi porozumením UML a znalosťou analýzy a návrhu softvéru, kedy čím vyššie porozumenie UML modelov účastníci uviedli, tým mali menšiu znalosť analýzy a návrhu softvéru. Z toho môžeme usúdiť, že účastníci radšej chápu UML modely, keďže v nich majú väčšie znalosti, ako analyzujú a navrhujú softvér vo všeobecnosti.

Negatívna korelácia medzi znalosťami UML a úspešnosťou riešenia otázok 1 a 3

môže súvisieť s tým, že študenti presne neodhadli svoje znalosti, teda sa im darilo menej, hoci uviedli vyššie znalosti UML modelovania. Otázka 1 pritom skúmala, či používatelia správne pochopili podmienku v tele cyklu, zatiaľ čo otázka 3 zisťovala, či si používatelia dostatočne zapamätali obsah diagramu. Predošlá znalosť AnimArchu výrazne nekoreluje s úspešnosťou riešenia jednotlivých otázok, a teda sa študentom darilo rovnako, bez ohľadu na ich predošlé skúsenosti s AnimArchom.

Jednotlivé otázky SUS dotazníku medzi sebou výrazne korelujú, či už negatívne alebo pozitívne. Najväčšie negatívne korelácie sú medzi SUS otázkami 4 a 7, kde sa ukázalo, že čím menej účastníci cítili potrebu pomoci technicky zdatnejšej osoby pri používaní AnimArchu, tým viac sa prikláňali k názoru, že väčšina ľudí by sa tento systém naučila používať veľmi rýchlo. Taktiež výrazne negatívne korelujú otázky 7 a 10, kedy sa ukázalo, že čím viac si účastníci mysleli, že väčšina ľudí sa naučí AnimArch používať rýchlo, tým menej nových vecí sa sami museli naučiť predtým, než ho začali používať. Z toho tiež vyplýva výrazná korelácia medzi otázkami 4 a 10 zo SUS dotazníka, čo naznačuje, že čím menej účastníci potrebovali pomoc technicky zdatnejšej osoby, tým menej nových vecí sa museli naučiť, pred samotným používaním AnimArchu. Z toho môžeme usúdiť, že použitie diagramu aktivít v AnimArchu bolo pre účastníkov ľahko pochopiteľné a nevyžadovalo si veľké predchádzajúce znalosti ani technickú podporu.

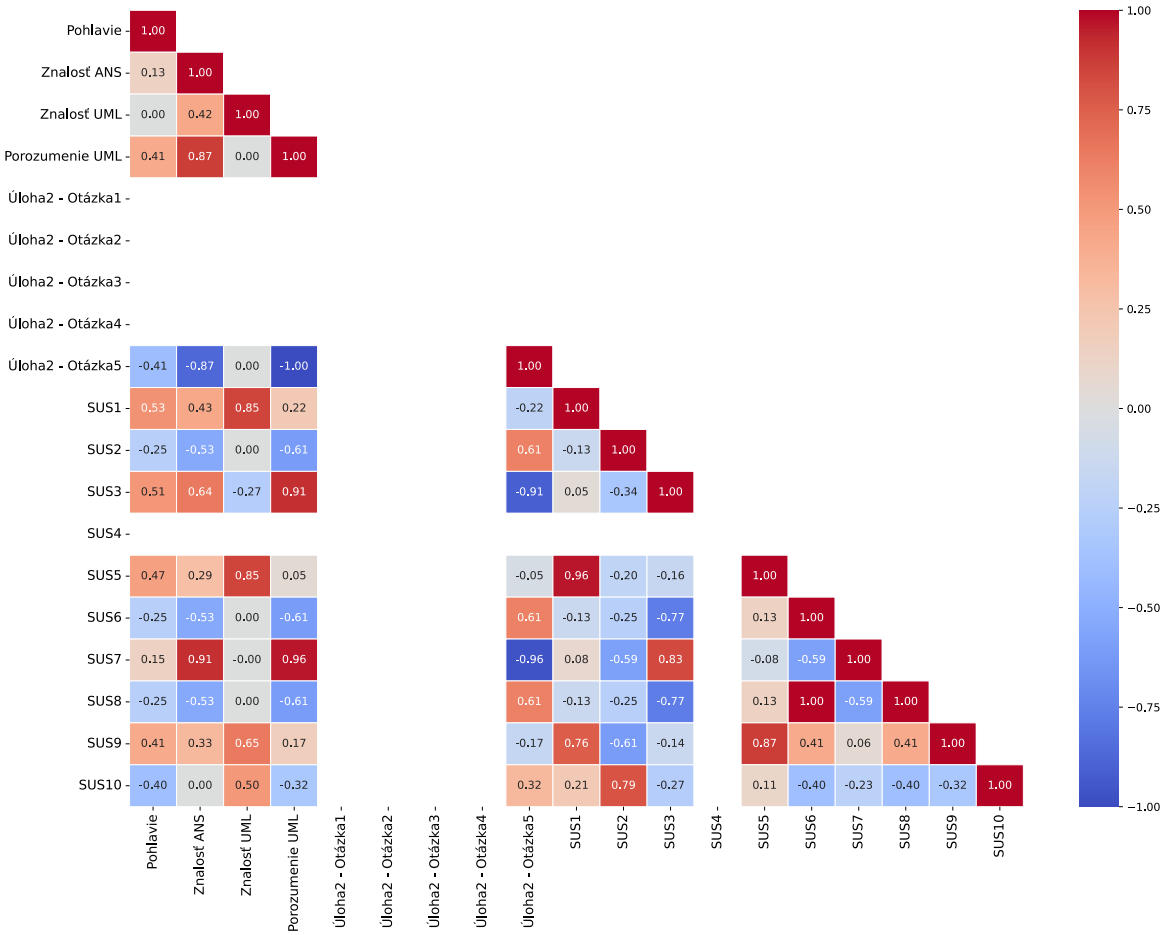
Pomerne výrazná negatívna korelácia sa vyskytuje aj v porovnaní mentálnej náročnosti úlohy a úspešnosťou riešenia otázok 1 a 3. To naznačuje, že používatelia, ktorí uviedli nízku mentálnu záťaž boli pri riešení otázok 1 a 3 úspešnejší. Taktiež výrazne koreluje fyzická náročnosť pri riešení úlohy a SUS otázky 4 a 10 a negatívne koreluje s otázkou 7 zo SUS dotazníka. Nízka fyzická náročnosť opäť súvisí s tým, že účastníci pociťovali nízku potrebu technicky zdatnejšej osoby, mysleli si, že väčšina ľudí sa naučí AnimArch používať rýchlo a bez potreby učenia nových vecí. Korelácia medzi tým, ako účastník ohodnotil svoju úspešnosť, a jeho znalosťou AnimArchu môže naznačovať, že účastníci, ktorí uviedli vyššie znalosti AnimArchu ohodnotili svoju úspešnosť horšie. Keďže je ale možné, že niektorí účastníci nepochopili správne hodnotenie svojej úspešnosti, túto vlastnosť nevieme presne interpretovať. Vysoká korelácia medzi tým, ako účastníci ohodnotili frustráciu počas riešenia úlohy a otázkou 2 zo SUS dotazníka súvisí s tým, že používatelia pociťovali nízku frustráciu a zároveň si mysleli, že systém nie je zbytočne zložitý. Z toho môžeme usúdiť, že animovaný diagram aktivít v AnimArchu sa účastníkom používal ľahko, a teda bol dobre navrhnutý a implementovaný.

Na obrázku 5.11 je znázornená korelačná teplotná mapa, zachytávajúca vzťahy medzi úspešnosťou riešenia úlohy 2, keď účastníci využívali statický diagram aktivít, a demografickými údajmi a odpoveďami na SUS dotazník.

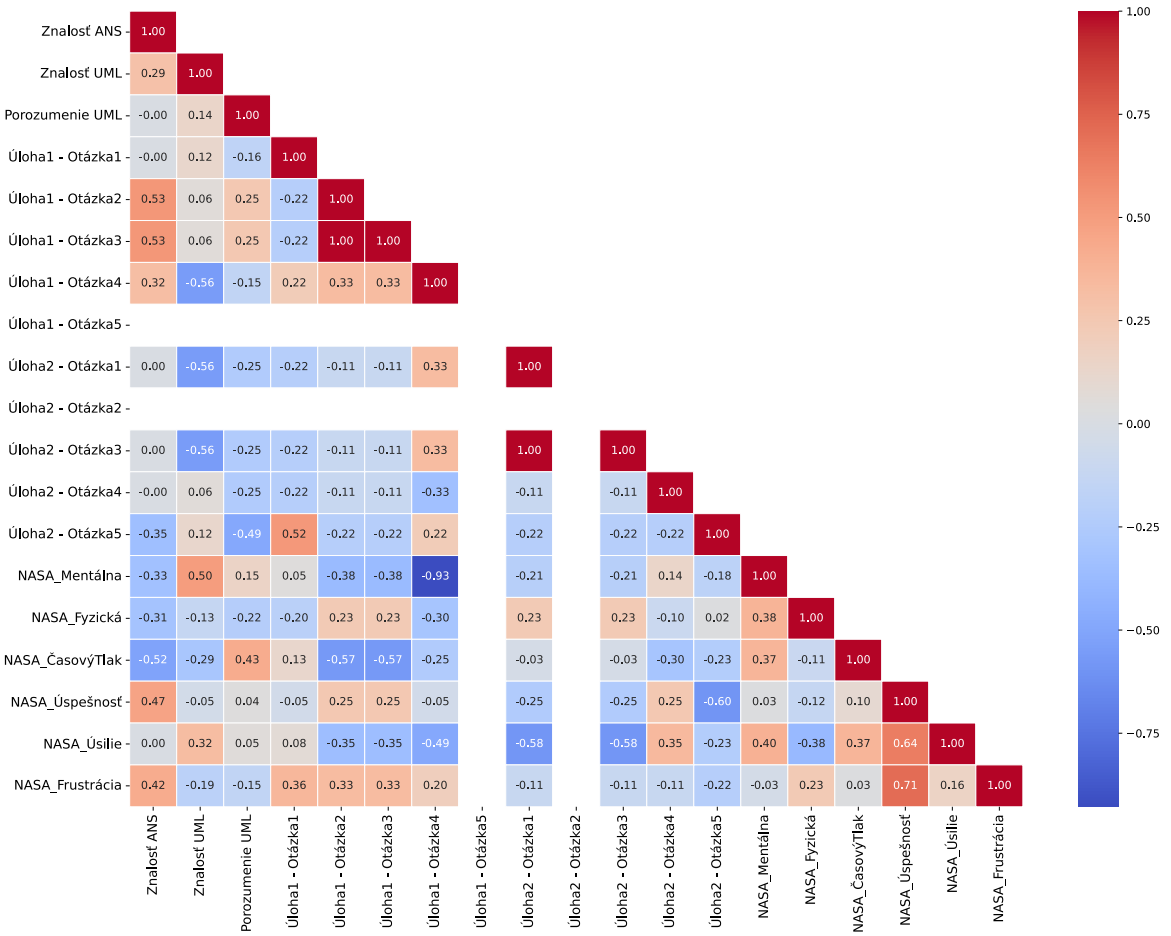
Úspešnosť riešenia otázky 5 negatívne koreluje s porozumením UML modelov, čo znamená, že účastníkom sa darilo menej, hoci uviedli vyššie porozumenie UML modelov. Piata otázka pritom skúmala, ako účastníci pochopili a zapamätali si daný diagram

aktivít. Môžeme teda usúdiť, že účastníci precenili svoje schopnosti chápania UML modelov.

Otázka 3 zo SUS dotazníka je v korelácii s porozumením UML modelov, kedy účastníci, ktorí ohodnotili, že systém sa ľahko používa, uviedli aj lepšie porozumenie UML modelov. Táto otázka tiež negatívne koreluje s úspešnosťou riešenia otázky 5, kedy sa účastníkom darilo horšie, hoci uviedli, že systém sa používa ľahko. SUS otázka 7 významne koreluje so znalosťami analýzy a návrhu softvéru a porozumením UML modelov. To značí, že účastníci, ktorí predpokladali, že väčšina ľudí by sa naučila systém ľahko používať mali aj väčšie znalosti z analýzy a návrhu softvéru a porozumenia UML modelov. Táto otázka je tiež v negatívnej korelácii s otázkou 1 zo SUS dotazníka, kedy účastníci uviedli, že by menej radi používali tento systém do budúcna, hoci si mysleli, že väčšina ľudí by sa naučila ho ľahko používať. Z toho môže vyplývať, že používatelia neboli dostatočne spokojní so statickým zobrazením diagramu aktivít, aj keď sa používal ľahko. Korelácia SUS otázky 6 a 8 naznačuje, že účastníci, ktorí považovali systém za menej ucelený, zároveň určili, že sa ovláda ťažšie. Z toho môžeme usúdiť, že statický diagram aktivít pôsobil na účastníkov nekonzistentne, a preto sa im ťažšie používal.



Obr. 5.11: Korelačná teplotná mapa znázorňujúca vzťahy medzi úspešnosťou riešenia úlohy 2 pomocou statického diagramu a zvyšnými dátami z dotazníku.



Obr. 5.12: Korelačná teplotná mapa znázorňujúca vzťahy medzi úspešnosťou riešenia oboch úloh a odpoveďami na NASA-TLX otázky.

Na obrázku 5.12 je znázornená korelačná teplotná mapa vzťahov medzi úspešnosťami riešenia oboch úloh a demografickými údajmi a odpoveďami na NASA-TLX dotazník.

Medzi úspešnosťou riešenia otázok 2 a 3 v úlohe 1 a znalosťou analýzy a návrhu softvéru je stredne silná korelácia. To naznačuje, že čím vyššiu znalosť analýzy a návrhu účastníci uvideli, tým lepšie sa im darilo. Naopak medzi úspešnosťami riešenia otázky 4 v úlohe 1, otázky 1 a otázky 3 v úlohe 2 a znalosťami UML modelov je stredne silná negatívna korelácia. To znamená, že účastníci boli v riešení týchto otázok menej úspešní, hoci uviedli vyššie znalosti UML modelov. Z toho môžeme usúdiť, že niektorí účastníci nevedeli správne odhadnúť mieru svojich znalostí UML modelov.

Výrazná negatívna korelácia je medzi pocitom mentálnej záťaže pri riešení úlohy a úspešnosťou riešenia otázky 4 v úlohe 1, kedy používatelia, ktorí uviedli vyššiu mentálnu náročnosť, boli aj menej úspešní. Pocit časového nátlaku tiež stredne negatívne koreluje so znalosťami analýzy a návrhu softvéru a úspešnosťami riešenia otázok 2 a 3 v úlohe 1. To naznačuje, že čím nižší časový nátlak účastníci pociťovali, tým vyššie znalosti analýzy a návrhu mali, a tiež boli úspešnejší pri riešení otázok. Z toho môžeme

usúdiť, že účastníci s väčšími znalosťami analýzy a modelovania boli pri riešení viac sebavedomí a efektívni, a preto pociťovali menší časový tlak.

5.2.6 Diskusia

Účastníci vo všeobecnosti hodnotili úlohy, ktoré riešili pomocou animovaného diagramu aktivít v AnimArchu, ako málo fyzicky náročné. Zároveň tiež pociťovali nízku frustráciu, nepociťovali potrebu pomoci technicky zdatnejšej osoby a mysleli si, že väčšina ľudí by sa naučila animovaný diagram aktivít v AnimArchu používať rýchlo a bez potreby učenia nových vecí.

Niektorí účastníci uviedli, že systém sa im zdal menej ucelený a museli sa pred použitím naučiť viac vecí. To mohlo byť spôsobené tým, že mali vyhradený pre nich príliš krátky čas na to, aby si plnohodnotne osvojili animáciu diagramu aktivít v AnimArchu, a preto sa im mohol zdať systém menej ucelený. Účastníci, ktorí pociťovali vyšší časový nátlak, uviedli, že nemali potrebu pomoci technicky zdatnejšej osoby. Z toho môžeme usúdiť, že účastníkom prišlo použitie AnimArchu jednoduché, keďže nepotrebovali pomoc technickejšej osoby, avšak by potrebovali viac času na osvojenie si systému, vďaka čomu by pracovali efektívnejšie, a teda by pociťovali nižší časový nátlak.

Účastníci, ktorí používali AnimArch pri riešení úlohy 1 boli úspešnejší, ak mali s AnimArchom predošlé skúsenosti. To môže byť spôsobené tým, že sa v softvéri cítili pohodlnejšie, keďže ho viac poznali. Účastníci, ktorí mali s AnimArchom menšie skúsenosti, mohli tiež sústrediť svoju pozornosť aj na zoznámenie sa so softvérom a nevenovali toľko pozornosti samotnému diagramu aktivít, a preto sa im darilo horšie. Pri riešení úlohy 2 pomocou AnimArchu sa ukázalo, že predošlé skúsenosti s AnimArchom nemali žiadny vplyv na úspešnosť riešenia úlohy.

Vo všeobecnosti teda môžeme usúdiť, že účastníkom sa narábalo s animovaným diagramom aktivít jednoducho, nepociťovali frustráciu ani potrebu pomoci technicky zdatnejšej osoby. Keďže sa jedná o nový spôsob vizualizovania diagramu aktivít prostredníctvom animovania v AnimArchu, s ktorým účastníci nemali doterajšie skúsenosti, na plnohodnotné osvojenie tohto spôsobu by potrebovali viac času. Aj napriek tomu ohodnotili animovaný diagram aktivít v AnimArchu ako dobre použiteľný, a teda môžeme tvrdiť, že naše riešenie sme navrhli a implementovali vhodne.

Závery sú vyvodzované len z dostupných dát, ktorých výpovedná hodnota môže byť obmedzená, vzhľadom na počet účastníkov testovania.

Záver

Do softvéru AnimArch sme implementovali vizualizovanie a animovanie diagramu aktivít. Ten sa generuje z načítaného kódu OAL, ktorý používateľ zvolí na začiatku animácie. Vďaka animácii sa v diagrame zvýrazňujú aktuálne vykonávané príkazy, čo môže prispieť k jednoduchšiemu pochopeniu vykonávaného kódu, najmä pri zložitejších procesoch. Implementovali sme tiež zobrazovanie viacerých diagramov aktivít za sebou. V prípade, že sa počas vykonávania zavolá nová metóda, vytvorí sa nový diagram aktivít, ktorý sa zobrazuje za pôvodným diagramom aktivít, čím vytvárame pomyselný zásobník volaní.

Naše riešenie sme evaluovali pomocou používateľského testovania. Testovania sa zúčastnilo 10 účastníkov, ktorí riešili dve úlohy pomocou rôznych nástrojov. Polovica účastníkov riešila prvú úlohu pomocou animovaného diagramu aktivít v AnimArchu a druhú úlohu riešila pomocou statického diagramu aktivít. Druhá polovica postupovala v opačnom poradí, a teda najskôr využili statický diagram a potom animovaný diagram aktivít. Z testovania sa ukázalo, že obom skupinám sa darilo približne rovnako dobre, bez výrazného rozdielu v úspešnosti riešení. Z výsledkov tiež môžeme určiť, že systém sa im používal ľahko, avšak by potrebovali viac času, aby si systém plnohodnotne osvojili.

Účastníci tiež hodnotili použiteľnosť systému, kde výsledné SUS skóre malo hodnotu 91.25, čo sa považuje za výbornú použiteľnosť. Zisťovali sme tiež záťaž účastníkov pri riešení úloh. Účastníci pociťovali nízku fyzickú záťaž a frustráciu, čo môže naznačovať, že sa im s animovaným diagramom aktivít v AnimArchu pracovalo dobre.

V budúcnosti by sa mohlo implementovať aj animovanie diagramu aktivít pri paralelnom vykonávaní. Taktiež by sa mohlo uskutočniť testovanie s väčším počtom účastníkov, ktoré by mohlo poskytnúť cenné informácie, ako vylepšiť zobrazovanie diagramu aktivít, aby boli používatelia úspešnejší, oproti využitiu statického diagramu aktivít, a tiež rôzne iné poznatky, ktoré by účastníci navrhli pridať alebo upraviť. Do softvéru AnimArch by sa mohlo v budúcnosti tiež pridať animovanie ďalšieho UML diagramu, napríklad sekvenčného diagramu. V AnimArchu sa aktuálne zobrazuje diagram tried, diagram objektov a naša práca implementovala zobrazenie diagramu aktivít. Vo výsledku sa teda animujú dva typy štrukturálnych UML diagramov a iba jeden behaviorálny UML diagram, preto by mohla pribudnúť práve vizualizácia sekvenčného diagramu, ktorý by zobrazoval ďalšie behaviorálne vlastnosti systému.

Literatúra

[1] Omer Salih and Abd-El-Kader Sahraoui. From Requirements Engineering to UML using Natural Language Processing – Survey Study. *European Journal of Engineering Research and Science*, 01 2017.

[2] The Unified Modeling Language. Dostupné na `https://www.uml-diagrams.org`. [Citované: 2024-11-17].

[3] Clarifying concepts: MBE vs MDE vs MDD vs MDA. Dostupné na `https://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/`. [Citované: 2024-12-04].

[4] Executable UML. Dostupné na `https://abstractsolutions.co.uk/our-services/executable-uml/`. [Citované: 2024-10-20].

[5] Uwe Zdun, Carsten Hentrich, and Schahram Dustdar. Modeling process-driven and service-oriented architectures using patterns and pattern primitives. *TWEB*, 1, 09 2007.

[6] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., USA, 3 edition, 2003.

[7] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. VILLE – A Language-Independent Program Visualization Tool. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88*, page 151–159. Australian Computer Society, Inc., 11 2007.

[8] Enes Yigitbas, Simon Gorissen, Nils Weidmann, and Gregor Engels. Design and evaluation of a collaborative UML modeling environment in virtual reality. *Softw. Syst. Model.*, 22(5):1397–1425, November 2022.

[9] Lukas Gregorovic and Ivan Polasek. Analysis and design of object-oriented software using multidimensional uml. In *Proceedings of the 15th International Conference on Knowledge Technologies and Data-Driven Business*, i-KNOW '15, New York, NY, USA, 2015. Association for Computing Machinery.

[10] Matej Ferenc, Ivan Polasek, and Juraj Vincur. Collaborative modeling and visualization of software systems using multidimensional UML. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 99–103, 09 2017.

[11] Jakub Kucecka, Juraj Vincur, Peter Kapec, and Pavel Cicak. UML-based live programming environment in virtual reality. In *2022 Working Conference on Software Visualization (VISSOFT)*, pages 177–181, 10 2022.

[12] Project Technology Inc. *Object Action Language Reference Manual*, 2008. Dostupné na `http://www.ooatool.com/docs/OAL08.pdf`.[Citované: 2024-11-30].

[13] Frederick Brooks, Jr. No Silver Bullet Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, 04 1987.

[14] Mohammad Hossain. Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management. *International Journal For Multidisciplinary Research*, 09 2023.

[15] Marc I Kellner, Raymond J Madachy, and David M Raffo. Software process simulation modeling: Why? what? how? *Journal of Systems and Software*, 46(2):91–105, 1999.

[16] Nenad Medvidovic, David Rosenblum, David Redmiles, and Jason Robbins. Modeling software architectures in the Unified Modeling Language. *Medvidovic, N. and Rosenblum, D.S. and Redmiles, D.G. and Robbins, J.E. (2002) Modeling software architectures in the unified modeling language. ACM Transactions on Software Engineering and Methodology, 11 (1). pp. 2-57. ISSN 1049331X*, 11, 01 2002.

[17] Gregor Engels, Reiko Heckel, and Stefan Sauer. UML - A universal modeling language? In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets*, ICATPN'00, Berlin, Heidelberg, 10 2000. Springer-Verlag.

[18] Frank Truyen. The Fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture. *Cephas Consulting Corp*, 2006.

[19] David Ameller, Xavier Burgués, Dolors Costal, Carles Farré, and Xavier Franch. Non-functional requirements in model-driven development of service-oriented architectures. *Science of Computer Programming*, 168:18–37, 2018.

[20] Alberto Silva. Model-driven engineering: A survey supported by a unified conceptual model. *Computer Languages, Systems & Structures*, 20, 06 2015.

[21] Martin Fowler. Language Workbenches and Model Driven Architecture. Dostupné na `https://martinfowler.com/articles/mdaLanguageWorkbench.html`, 05 2005. [Citované: 2024-12-04].

[22] Marc Balcer and Ivar Jacobson. Executable UML: A Foundation for Model-Driven Architectures. 01 2002.

[23] Miguel Luz and Alberto Rodrigues da Silva. Running and Debugging UML Models. Lisboa, Portugal, 2004. INESC-ID.

[24] Federico Ciccozzi, Ivano Malavolta, and Bran Selic. Execution of UML models: a systematic review of research and practice. *Software & Systems Modeling*, 18(3):2313–2360, 06 2019.

[25] BridgePoint IDE. Dostupné na `https://xtuml.org/`. [Citované: 2024-12-01].

[26] Keith E. Brown. Navigating the rover with xtUML. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018.

[27] Martin Siebenhaller and Michael Kaufmann. Drawing activity diagrams. In *Proceedings of the 2006 ACM Symposium on Software Visualization*, SoftVis '06, page 159–160, New York, NY, USA, 2006. Association for Computing Machinery.

[28] Lukas Radosky and Ivan Polasek. Executable multi-layered software models. In *Proceedings of the 1st International Workshop on Designing Software*, Designing '24, page 46–51, New York, NY, USA, 2024. Association for Computing Machinery.

[29] James R. Lewis and Jeff Sauro. Item benchmarks for the system usability scale. *J. Usability Studies*, 13(3):158–167, May 2018.

[30] Anjana Ramkumar, Pieter Jan Stappers, W.J. Niessen, Sonja Adebahr, Tanja Schimek-Jasch, Ursula Nestle, and Wolf Song. Using goms and nasa-tlx to evaluate human-computer interaction process in interactive segmentation. *International Journal of Human-Computer Interaction*, 33:1–12, 09 2016.

Príloha A: Používateľské testovanie

Táto príloha obsahuje úlohy, ktoré riešili účastníci testovania.

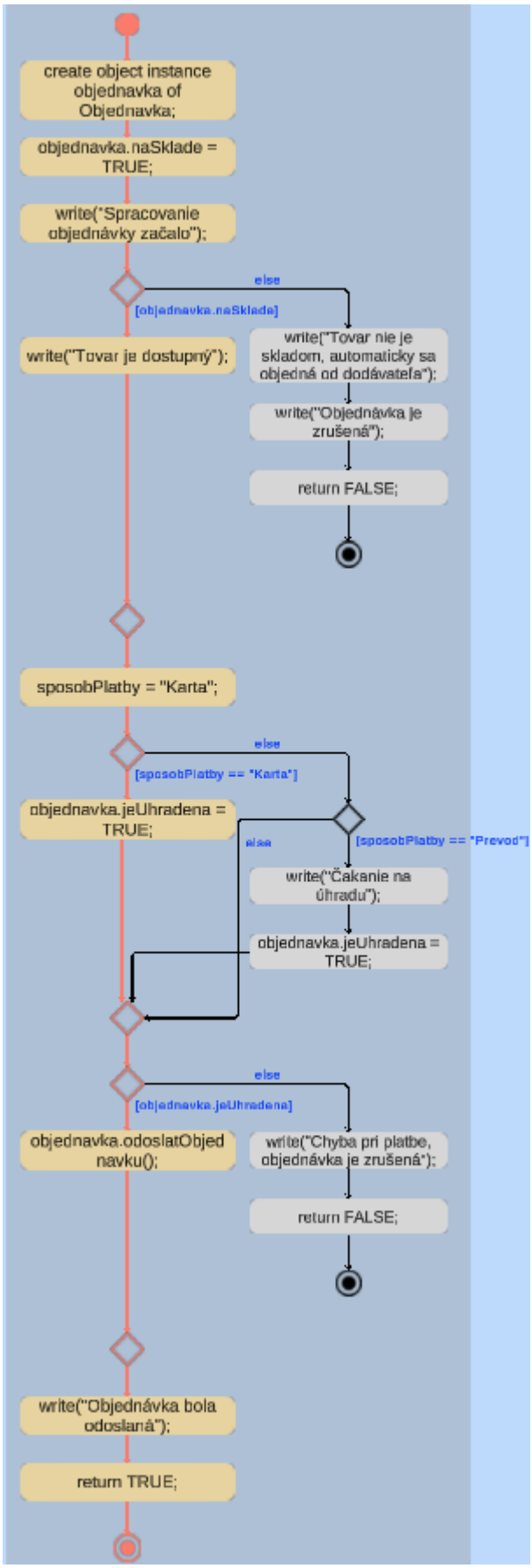
Úloha 1

V úlohe 1 účastníci využívali niektorý z diagramov zobrazených na obrázku 5.13 a 5.14. Po preštudovaní diagramu účastníci odpovedali na nasledovné otázky:

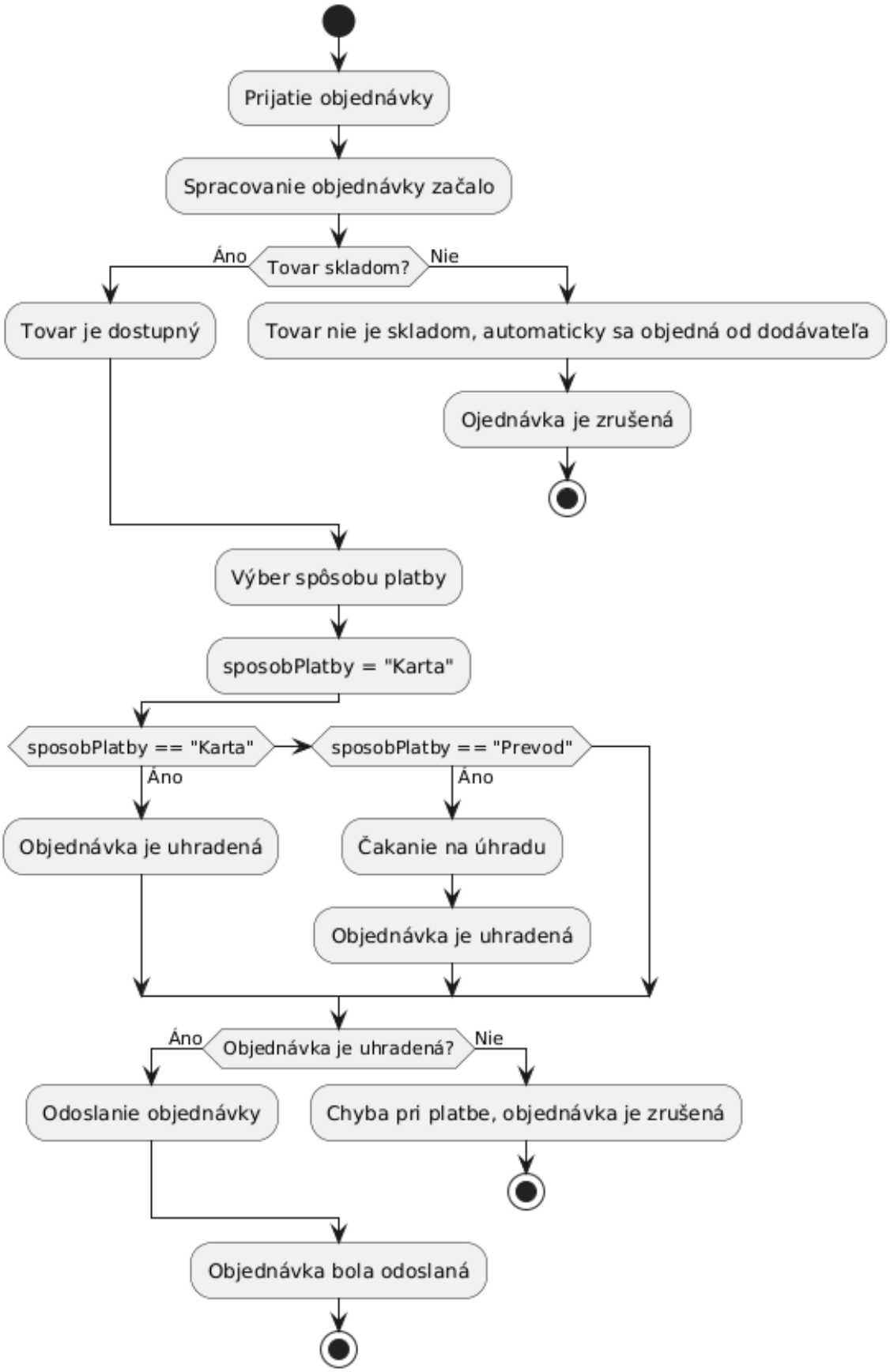
1. Čo sa stane, ak tovar nie je skladom?
    - Proces pokračuje k výberu spôsobu platby.
    - Objednávka je zrušená.
    - Systém automaticky objedná tovar od dodávateľa.
    - Objednávka sa odloží na neskôr.
2. Aké možnosti platby sú akceptované podľa diagramu?
    - Karta, Prevod
    - Dobierka, Karta
    - Karta, Hotovosť
    - Hotovosť, Dobierka
3. Čo sa stane, ak platba nie je úspešná?
    - Objednávka pokračuje bez platby.
    - Používateľ je požiadaný o opravu údajov.
    - Používateľ je informovaný a objednávka je zrušená.
    - Používateľ je presmerovaný na dobierku.
4. Ktorý krok nasleduje po úspešnej platbe?
    - Informovanie používateľa o stave objednávky.
    - Odoslanie objednávky.

- Zrušenie objednávky.
- Presmerovanie používateľa na hlavné menu.

5. Aké podmienky musia byť splnené, aby objednávka mohla byť odoslaná?
   - Tovar musí byť na sklade a platba musí byť úspešná.
   - Tovar musí byť na sklade a platba musí byť vykonaná kartou.
   - Platba musí byť úspešná, ale dostupnosť tovaru nie je dôležitá.
   - Objednávka môže byť odoslaná bez ohľadu na dostupnosť tovaru.



Obr. 5.13: Animovaný diagram aktivít v AnimArchu pre úlohu 1.

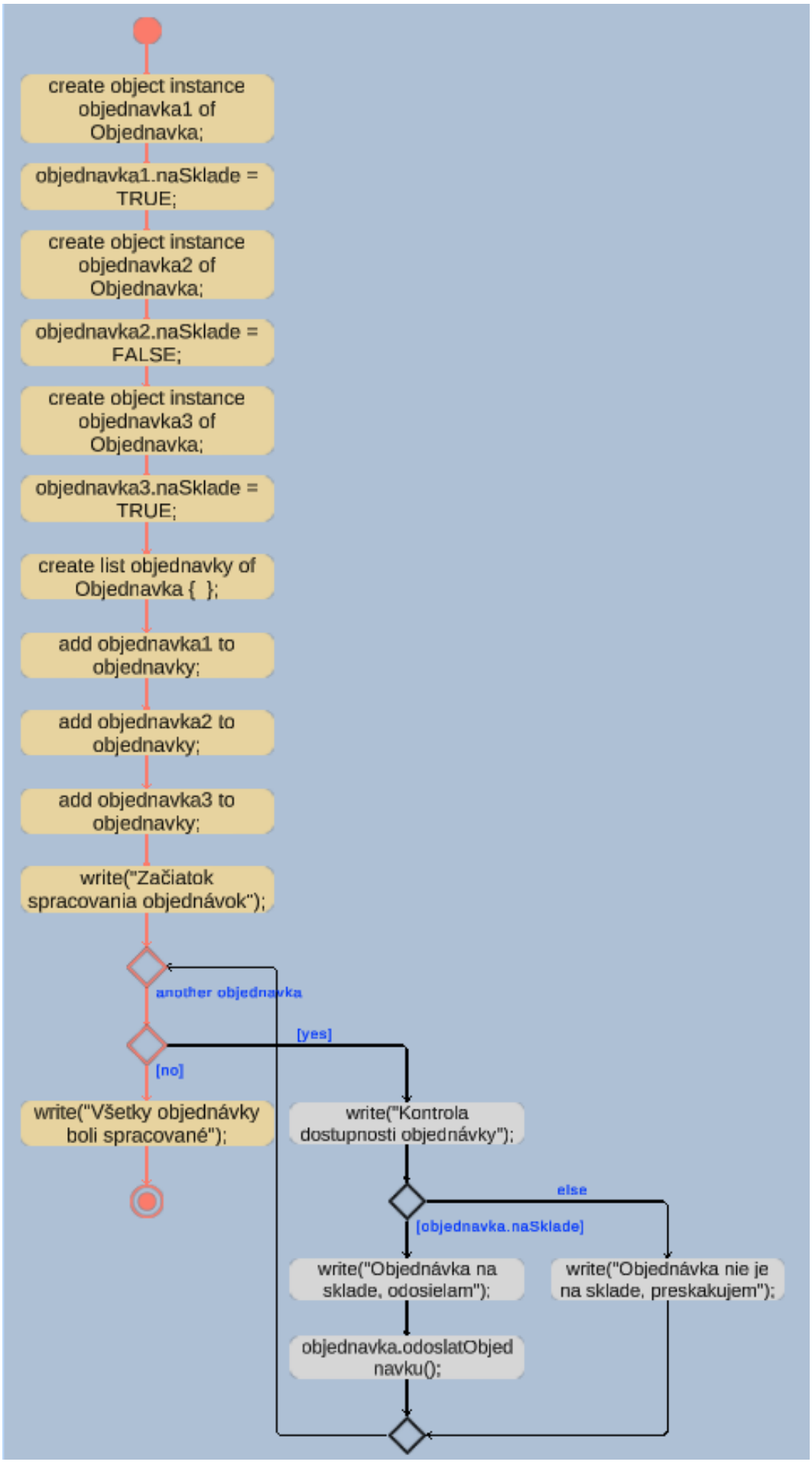


Obr. 5.14: Statický diagram aktivít pre úlohu 1.

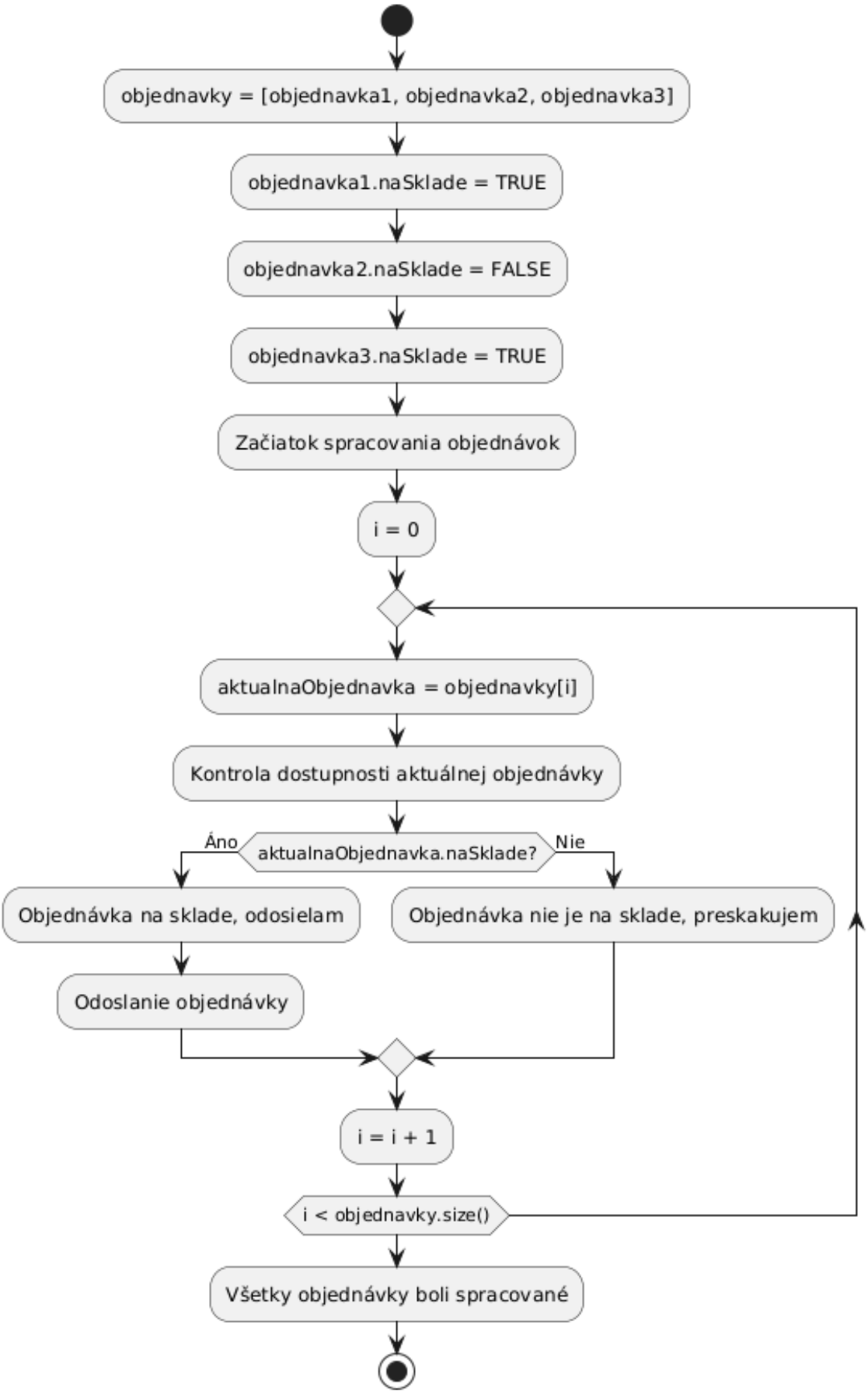
Úloha 2

V úlohe 2 účastníci využívali niektorý z diagramov zobrazených na obrázku 5.15 a 5.16. Po preštudovaní diagramu ďalej odpovedali na nasledovné otázky:

1. Čo sa stane, ak objednávka nie je na sklade?
   - Objednávka sa spracuje a odošle.
   - Objednávka sa preskočí.
   - Objednávka sa odstráni zo zoznamu.
   - Objednávka sa označí ako vybavená.
2. Koľko objednávok sa vytvorí v metóde vytvorObjednavky?
   - 1
   - 2
   - 3
   - 4
3. Aký je stav atribútu naSklade pre objednavka2?
   - TRUE
   - FALSE
   - NULL
   - Nie je definovaný.
4. Čo sa vykoná pre objednávku, ktorá je na sklade?
   - Objednávka sa preskočí.
   - Objednávka sa odošle.
   - Objednávka sa odstráni zo zoznamu.
   - Objednávka sa označí ako nevybavená.
5. Čo sa stane po spracovaní všetkých objednávok?
   - Zoznam objednávok sa vymaže.
   - Systém oznámi, že všetky objednávky boli spracované.
   - Objednávky sa znova spracujú.
   - Proces sa ukončí bez výstupu.



Obr. 5.15: Animovaný diagram aktivít v AnimArchu pre úlohu 2.



Obr. 5.16: Statický diagram aktivít pre úlohu 2.

Príloha B: Elektronická príloha

Elektronická príloha obsahuje zdrojový kód aplikácie.
Zdrojový kód sa nachádza aj online na adrese `https://github.com/karkub/AnimArch`.