Ekonomická fakulta
Faculty of Economics

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované matematiky a informatiky

Bakalářská práce

Vývoj aplikace pro plánování projektů pomocí metody síťové analýzy

Vypracoval: Jan Pech
Vedoucí práce: Mgr. Radim Remeš

České Budějovice 2017

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta ekonomická
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan PECH**
Osobní číslo:	**E14377**
Studijní program:	**B6209 Systémové inženýrství a informatika**
Studijní obor:	**Ekonomická informatika**
Název tématu:	**Vývoj aplikace pro plánování projektů pomocí metody síťové analýzy**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování:

Cílem práce je vytvořit aplikaci pro plánování projektů metodou síťové analýzy.

Metodický postup:

1. Studium odborné literatury.
2. Publikace výsledků rešerše.
3. Návrh, popis vývoje a implementace aplikace.
4. Zhodnocení, vypracování doporučení a závěrů.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 - 50 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **Hillier, F. S., & Lieberman, Gerald J. (2014).** *Introduction to Operations Research.* **Boston: McGraw-Hill.**
2. **Jablonský, J. (2007).** *Operační výzkum: kvantitativní metody pro ekonomické rozhodování.* **3. vyd. Praha: Professional.**
3. **Microsoft. (2015). Microsoft Solver Foundation 3.1. Microsoft Developer Network: MSDN Library [online]. 2016.03.15 [cit. 2016-03-29]. Dostupné z: <https://msdn.microsoft.com/en-us/library/ff524509>.**
4. **Nathan, A. (2013).** *WPF 4.5 Unleashed.* **Indianapolis, IN, USA: Sams.**
5. **Troelsen, A., & Japikse, P. (2015).** *C# 6.0 and the .NET 4.6 Framework.* **7. vydání. New York, USA: Apress.**

Vedoucí bakalářské práce: **Mgr. Radim Remeš**

Katedra aplikované matematiky a informatiky

Konzultant bakalářské práce: **RNDr. Jana Klicnarová, Ph.D.**

Katedra aplikované matematiky a informatiky

Datum zadání bakalářské práce: **15. ledna 2016**

Termín odevzdání bakalářské práce: **14. dubna 2017**

doc. Ing. Ladislav Rolínek, Ph.D.
děkan

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA
Studentská 13 (26)
370 05 České Budějovice

L.S.

prof. RNDr. Pavel Tlustý, CSc.
vedoucí katedry

V Českých Budějovicích dne 31. března 2016

Prohlášení

Prohlašuji, že svoji bakalářskou práci na téma „Vývoj aplikace pro plánování projektů pomocí metody síťové analýzy“ jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47 zákona č. 111/1998 Sb. V platném znění souhlasím se zveřejněním své bakalářské práce, a to – v nezkrácené podobě/v úpravě vzniklé vypuštěním vyznačených částí archivovaných Ekonomickou fakultou - elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou u Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

Datum	Podpis studenta

PODĚKOVÁNÍ

Největší poděkování patří vedoucímu mé bakalářské práce Mgr. Radimu Remešovi, kterému tímto děkuji za vstřícnost, ochotu, trpělivost a věcné připomínky, které velkou mírou přispěly ke zdárnému dokončení této práce. Dále bych chtěl poděkovat své rodině za veškerou podporu, kterou během mých studií projevovala a projevuje. Nakonec bych rád poděkoval všem svým blízkým přátelům (nejvíce Anně Prokopové), kteří při mně stáli během těžkých chvil, které mě při mém studiu postihly.

Obsah práce

1 Úvod

Jedním z nejtěžších úkolů, kterému manažer může čelit, je řízení rozsáhlého projektu, který se skládá z velkého počtu navzájem navazujících aktivit, které je potřeba koordinovat. Tento proces se nazývá řízení projektů (anglicky project management). Za slovem „projekt“ si můžeme představit vyvinutí úsilí za účelem dosažení požadovaného výsledku (produkt, služba atd.). Často je projekt omezený různými skutečnostmi (časem, financemi aj.). Z toho vyplývá, že primárním úkolem řízení projektů je za daných omezení dopracovat se ke stanovenému výsledku. Sekundárním úkolem řízení projektů je celý projekt řádně optimalizovat (správná alokace a využití zdrojů apod.).

Historie řízení projektů sahá až do začátků 19. století, kdy se jím zabýval Henry Gantt, jehož diagram (Ganttův diagram) se používá pro vizualizaci projektů i v dnešní době. Vznikaly různé techniky, které se postupem času vylepšovaly a na jejich myšlenkách vznikaly metody nové. V 50. letech 20. století nezávisle na sobě vznikly postupy pro řízení projektů CPM a PERT, které se využívají do dnes, a o kterých pojednává tato práce. V roce 1967 bylo založeno nadnárodní sdružení IPMA (International Project Management Association), jehož cílem je standardizovat postupy a znalosti vztahující se k řízení projektů. Kromě CPM a PERT existují i jiné techniky využívané k řízení projektů (PRINCE2, Critical chain project management, Project production management atd.), které částečně vycházejí z CPM a PERT a snaží se odstranit jejich nedostatky.

1.1 Cíl práce

Cílem této práce je seznámit se s metodami CPM a PERT, které mají uplatnění při řízení projektů a vyvinutí aplikace založené na těchto metodách. CPM a PERT poskytují analýzy zaměřené na různé oblasti této problematiky. Analýzou, kterou se zabývá největší část teoretické části práce, je tzv. časová analýza projektu. V praktické části je popsáno, jakým způsobem může být celá tato analýza řešena pomocí programovacího jazyka C# a zásad objektově orientovaného programování. Je zde vysvětlen návrh modelu tříd a následná implementace početních algoritmů popsaných v teoretické části. Dále je zde uveden postup tvorby vlastního souborového formátu, tvorba instalačního souboru aplikace a způsob ochrany zdrojového kódu před dekompilací pomocí tzv. obfuskace.

2 Řízení projektů pomocí PERT/CPM

PERT (Program Evaluation And Review Technique) a CPM (Critical Path Method) jsou techniky, které mají manažeři k dispozici při nesení rizik spojených s plánováním a řízením projektů. Tyto techniky jsou založené na znalostech z diskrétní matematiky (konkrétně teorie grafů). K vizualizaci projektů využívají tzv. síťových grafů. K dispozici je mnoho komerčních i nekomerčních aplikací, které jsou založené na těchto technikách a po zadání potřebných informací manažerům asistují s plánováním a následným monitorováním projektu. Nejznámější komerčně dostupnou aplikací je MS Project, kterou poskytuje firma Microsoft.

PERT a CPM se mohou využít k analýzám různých aspektů projektu. Nejvíce využívanými analýzami, které PERT a CPM poskytují, jsou časová analýza projektu (zaměřuje se na otázku „Jak dlouho projekt bude trvat?“), časově-zdrojová analýza (zabývá se správnou alokací dostupných zdrojů) projektu a časově-nákladová analýza projektu (řeší situace, kdy se může vyplatit vydat více zdrojů za dřívější dokončení projektu).

PERT a CPM byly nezávisle na sobě vymyšleny v pozdních 50. letech 20. století. Od té doby patří k nejvíce využívaným technikám pro řízení projektů. Originální verze obou technik mají oproti dnes využívaným verzím některé důležité rozdíly. Obě metody byly a jsou používány ve velmi různorodých a rozsáhlých projektech. Zde jsou pro představu některé druhy projektů, pro které byly využity (Hillier a Lieberman, 2001):

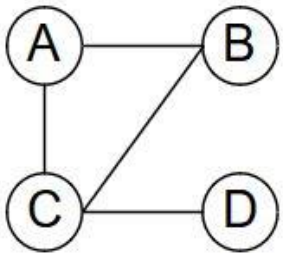
- Výstavba nové továrny
- Výzkum a vývoj nového produktu
- Vesmírné projekty NASA
- Produkce filmů
- Konstrukce lodi
- Státem financovaný projekt na vývoj nového zbraňového systému
- Přemístění významných zařízení
- Údržba nukleárního reaktoru
- Instalace informačního systému
- Řízení reklamní kampaně

Obě techniky jsou založené na stejných principech, a tudíž jsou si velmi podobné. Hlavní rozdíl je ve vnímání délky trvání jednotlivých aktivit, z kterých se projekt skládá. Jejich užití závisí na datech, která jsou k dispozici (ty rozhodnou o tzv. druhu projektu).

2.1 Síťový graf

Síťový graf (Obrázek 1) se skládá ze souboru bodů a linií propojujících určité páry bodů. Bodům se říká vrcholy (anglicky vertices) nebo uzly (anglicky nodes). Linie jsou nazývány hrany (anglicky edges).

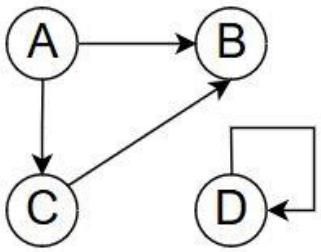
Obrázek 1 - Síťový graf



(zdroj: Autor)

Hrany grafu mohou mít směr, který se znázorní šipkou. Taková hrana se pak označuje jako orientovaná. Když se v grafu nacházejí jen orientované hrany, tak je graf označen jako orientovaný (Obrázek 2).

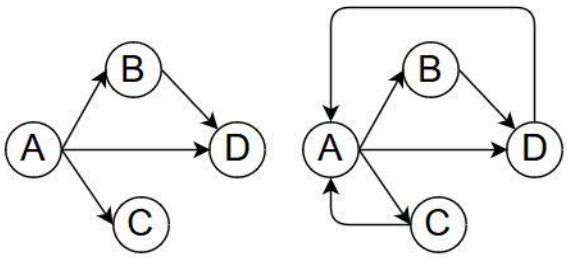
Obrázek 2 - Orientovaný graf



(zdroj: Autor)

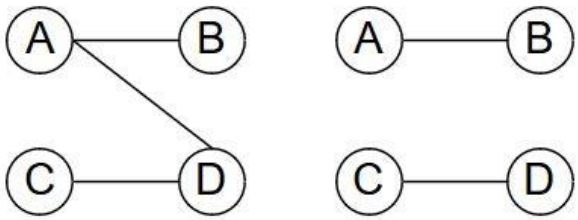
Když vrcholy grafu nejsou propojeny pomocí hrany, tak nastává otázka, jestli k jejich propojení nedojde pomocí sérií hran. Cesta mezi dvěma vrcholy je sekvence rozdílných hran propojujících tyto vrcholy. Graf je acyklický (opak grafu cyklického, Obrázek 3), když neobsahuje žádný cyklus (neexistuje cesta, kterou se lze vydat pryč z vrcholu a následně se do něj zpátky vrátit). Pokud pro graf (Obrázek 4) platí, že pro všechny vrcholy grafu existuje alespoň jedna cesta, která je spojuje, tak se nazývá souvislý (opakem je graf nesouvislý).

Obrázek 3- Acyklický a cyklický graf



(zdroj: Autor)

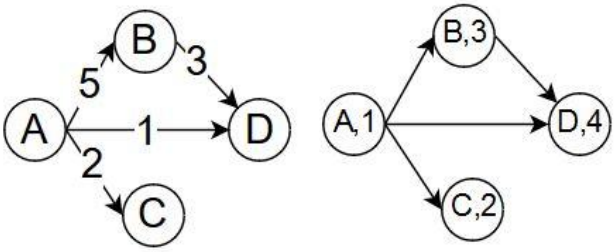
Obrázek 4 - Souvislý a nesouvislý graf



(zdroj: Autor)

Vrcholům nebo hranám lze přiřadit číselné hodnoty, které například mohou znázorňovat vzdálenost těchto dvou hran. Graf je pak označen jako ohodnocený.

Obrázek 5 - Ohodnocený graf pomocí hran a ohodnocený graf pomocí vrcholů



(zdroj: Autor)

Využití síťových grafů je velmi variabilní a jejich užití lze nalézt i v procesech, které lidé používají denně:

- The Shortest-Path Problem - hledání nejkratší cesty v grafu, využití př. IDOS, Google Maps
- The Minimum Spanning Tree Problem – využití např. při návrhu telekomunikačních sítí nebo návrhu rozmístění vedení vysokého napětí
- The Maximum Flow Problem – využití např. pro nalezení maximálního využití zásobovací sítě společnosti

2.2 Použití síťového grafu k vizualizaci projektu

Acyklický orientovaný ohodnocený lze využít k vizualizaci projektu, v kterém vrcholy znázorňují jednotlivé aktivity (zapisuje se do nich jméno či zkratka aktivity a odhadovaná délka trvání) a hrany reprezentují závislosti mezi jednotlivými aktivitami. Tomuto přístupu k vizualizaci projektu se říká AON (activity-on-node). Existuje druhý způsob ke znázornění projektu a to AOA (activity-on-arc), ve kterém jsou aktivity znázorněny pomocí hran a vrcholy jsou zde k oddělení aktivit. Originální verze PERT a CPM využívali AOA přístup, ale postupem času se přešlo k AON přístupu, jelikož takový graf je daleko lehčí sestavit, lépe mu rozumí uživatelé neznající PERT a CPM a snáze se zachycují změny v projektu. K sestavení grafu jsou potřeba 3 druhy informací (Hillier a Lieberman, 2001):

- Informace o aktivitách – rozebrání projektu do jednotlivých aktivit (do požadované úrovně detailnosti)
- Přednostní vztahy – identifikace předchůdců každé aktivity
- Časová informace – odhadovaná délka trvání každé aktivity

V tabulce 1 jsou zapsány informace o fiktivním projektu (přestavba a rekonstrukce kanceláří). Při vizualizaci projektu (Obrázek 6) by mohly vzniknout problémy (nebylo by na první pohled zřetelné, kterými aktivitami projekt začíná nebo končí, Obrázek 7), kdy by celý projekt nekončil nebo nezačínal jednou aktivitou. Z toho důvodu se přidají dvě fiktivní aktivity znázorňující začátek a konec projektu. Všem aktivitám, které nemají žádného předchůdce, se zadá jako předchůdce fiktivní aktivita znázorňující začátek projektu. Aktivity, které nemají žádného následovníka (ukončují projekt), se zadají jako předchůdci aktivitě znázorňující konec projektu.

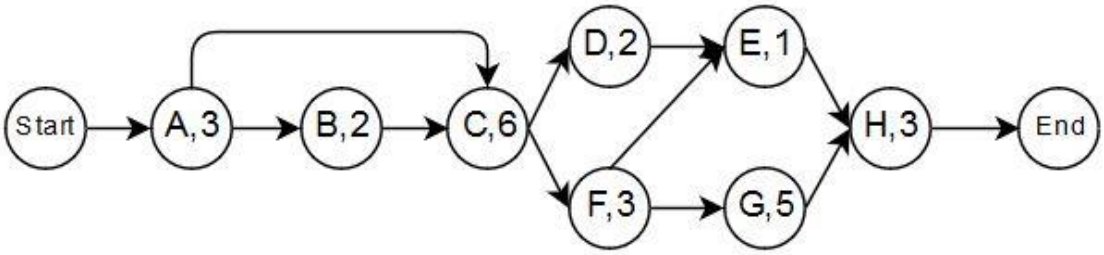
Tabulka 1 - Zadání projektu

Zkratka aktivity	Doba trvání (dny)	Předcházející aktivity	Popis
Start	0	-	Začátek projektu
A	3	Start	Vyklizení osobních věcí
B	2	A	Vystěhování počítačů a nábytku
C	6	A, B	Podlahy a vymalování

D	2	C	Nábytek
E	1	D, F	Síťové přípojky
F	3	C	Nové pracovní stoly
G	5	F	Vybavení HW
H	3	E, G	Instalace SW a napojení na síť
End	0	H	Konec projektu

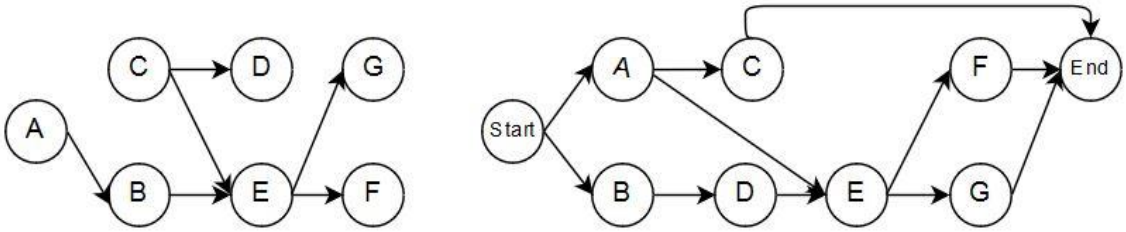
(zdroj: http://www2.ef.jcu.cz/~jfrieb/tspp/data/pripadove_studie/projekt.pdf)

Obrázek 6 - Síťový graf znázorňující přestavbu kanceláří



(zdroj: Autor)

Obrázek 7 - Vizualizace projektu s a bez užití fiktivních pomocných aktivit



(zdroj: Autor)

2.3 Časová analýza projektu

Hillier a Lieberman (2001) stanovili po zamyšlení nad projektem několik otázek, které jsou pro řízení projektu velmi důležité:

- Otázka 1: Jaká je celková doba trvání projektu, pokud se neobjeví žádné zpoždění?
- Otázka 2: Kdy jednotlivé aktivity musí začínat a končit (nejpozději), aby se projekt nezpozdil?
- Otázka 3: Kdy jednotlivé aktivity mohou začínat a končit (nejdříve), když se neobjeví žádné zpoždění?

- Otázka 4: Které aktivity nesmí mít zpoždění, aby se předešlo zpoždění celého projektu?
- Otázka 5: Kterým aktivitám může být tolerováno zpoždění a jaké?

Všechny výše uvedené otázky lze zodpovědět z vizualizace projektu (Obrázek 6), ale u větších projektů se v grafu lehce něco přehlédne. Proto je lepší využít početní algoritmus vysvětlený v následujících kapitolách.

2.3.1 Kritická cesta

Cesta projektem je série navzájem propojených aktivit, které začínají ve fiktivní aktivitě znázorňující začátek projektu a končí ve fiktivní aktivitě znázorňující konec projektu. Délka trvání cesty je ve většině případů součet odhadnutých dob trvání aktivit v cestě.

V projektu (Tabulka 1, Obrázek 6) se nachází 6 cest:

- Cesta 1: Start→A→B→C→F→G→H→End (suma dob trvání aktivit: 22)
- Cesta 2: Start→A→B→C→D→E→H→End (suma dob trvání aktivit: 17)
- Cesta 3: Start→A→B→C→F→E→H→End (suma dob trvání aktivit: 18)
- Cesta 4: Start→A→C→F→G→H→End (suma dob trvání aktivit: 20)
- Cesta 5: Start→A→C→D→E→H→End (suma dob trvání aktivit: 15)
- Cesta 6: Start→A→C→F→E→H→End (suma dob trvání aktivit: 16)

U cesty 1 je součet odhadnutých dob trvání aktivit, které se v ní nacházejí, 22, což je největší suma z nalezených cest. Tato cesta je proto označena jako kritická a její délka odpovídá délce projektu. To znamená, že projekt bude trvat 22 dnů. Aktivity na této cestě se nesmí zpozdit, jelikož by to znamenalo, že se zpozdí celý projekt.

Cesta 3 má součet odhadnutých dob trvání aktivit, které se v ní nacházejí 18. Oproti kritické cestě je zde rozdíl, že místo aktivity G se v ní nachází aktivita E. Obě tyto aktivity předcházejí aktivitě H. Aktivita E trvá 1 den a G 5 dnů. To znamená, že aktivita E má časovou rezervu 4 dny (jedná se o volnou časovou rezervu, což je vysvětleno později v této práci). Tato cesta však bude trvat také 22 dnů, protože i když dokončíme aktivity E v nejdřívějším možném čase, tak aktivita H musí počkat na dokončení aktivity G. Cesta 3 proto není označena jako kritická, jelikož aktivita v ní má časovou rezervu.

Cesta 2 je podobný případ jako cesta 3. Opět obsahuje aktivity, které mají časovou rezervu, ale bude čekat na dokončení aktivity H. Délka jejího trvání je tedy 22, ale není kritická (obsahuje aktivity, které mají časovou rezervu).

Cesta 4 se od cesty 1 liší v tom, že neobsahuje aktivitu B. Jelikož aktivita C však bude čekat na dokončení aktivity B, tak její délka není 20 dnů ale 22. Vztah, kdy aktivita A podmiňuje aktivity B a C a aktivita B podmiňuje C, se nazývá jako tranzitivní a závislost aktivity C na aktivitě A je zbytečná, protože aktivita C bude vždy čekat na aktivitu B, která je podmíněná aktivitou A. Vlastnosti této cesty jsou tedy totožné jako cesty 1. Mají stejné vlastnosti, je také kritická a jediný rozdíl je ve výše popsané tranzitivitě.

To stejné platí pro cestu 5 a cestu 6, kde cesta 5 má stejné vlastnosti jako cesta 2 a cesta 6 má stejné vlastnosti jako cesta 3. Shodují se ve všech vlastnostech (délka trvání, časové rezervy aktivit, obě nejsou kritické) a jediný rozdíl je opět v tranzitivitě u aktivit A, B a C.

Tímto způsobem se získaly odpovědi na otázky 1 a 4 položené na konci předchozího tématu. Pro menší projekty může stačit nakreslit si vizualizaci projektu (Obrázek 6), následně v ní nalézt všechny cesty a určit kritickou cestu. U větších projektů však může nastat problém. Proto PERT a CPM mají daleko efektivnější početní způsob, jak na tyto otázky odpovědět. Tento způsob je daleko efektivnější u větších projektů. Také poskytuje daleko více informací a rovnou odpoví na výše položené otázky (Hillier a Lieberman, 2001).

2.3.2 Plánování jednotlivých aktivit

Hillier a Lieberman (2001) říkají, že plánování jednotlivých aktivit začíná otázkou „Kdy jednotlivé aktivity mohou začínat a končit (nejdříve), když se neobjeví žádné zpoždění?“. To znamená, že každá aktivita bude začínat, co nejdříve to půjde. Bude se zjišťovat nejdřívější možný začátek aktivity (zkratka *ES*, anglicky earliest start time) a nejdřívější možný konec aktivity (zkratka *EF*, anglicky earliest finish time), které se vypočítají pomocí vztahů:

- $EF = ES + odhadovaná\ doba\ trvání\ aktivity$
- $ES = největší\ EF\ předchůdců$

Následuje ukázkový výpočet pro aktivity popsané v tabulce 1. Počítat se začíná u aktivity Start a přes její následovníky se postupuje směrem k aktivitě End. Výpočet pro aktivitu Start:

- $ES = 0$ $(aktivita\ Start\ má\ vždy\ ES\ rovno\ 0)$
- $EF = 0$ $(aktivita\ Start\ má\ vždy\ EF\ rovno\ 0)$

Výpočet pro aktivitu A:

- $ES = Maximum\ z\ (Start\ EF = 0) \rightarrow 0$
- $EF = 0 + 3 = 3$

Výpočet pro aktivitu B:

- $ES = Maximum\ z\ (A\ EF = 3) \rightarrow 3$
- $EF = 3 + 2 = 5$

Výpočet pro aktivitu C:

- $ES = Maximum\ z\ (A\ EF = 3; B\ EF = 5) \rightarrow 5$
- $EF = 5 + 6 = 11$

Stejným způsobem se zjistí *ES* a *EF* každé aktivity v projektu. Tyto výpočty je pro přehlednost vhodné zapisovat do tabulky (Tabulka 2).

Tabulka 2 - Výpočet ES a EF aktivit

Zkratka aktivity	Doba trvání (dny)	Předcházející aktivity	ES	EF
Start	0	-	0	0
A	3	Start	0	3
B	2	A	3	5
C	6	A, B	5	11
D	2	C	11	13
E	1	D, F	14	15
F	3	C	11	14
G	5	F	14	19
H	3	E, G	19	22
End	0	H	22	22

(zdroj: Autor)

Je třeba myslet na to, že plán, který se tímto postupem získá, počítá s tím, že doba trvání aktivity bude stejná, jako odhadnutá doba trvání. Co se stane, pokud nějaká aktivita bude mít zpoždění? Prodlouží to trvání projektu? To záleží na tom, která aktivita se zpozdí.

Další krok dle autorů Hillier a Lieberman (2001) je zjištění toho, kdy jednotlivé aktivity mohou nejdéle začít a skončit, aby se projekt nezpozdil. Zjišťuje se nejpozdější možný začátek aktivity (zkratka *LS*, anglicky latest start time) a nejpozdější možný konec aktivity (zkratka *LF*, anglicky latest finish start), které se spočítají pomocí vzorců:

- $LS = LF - odhadovaná\ doba\ trvání\ aktivity$
- $LF = nejmenší\ LS\ následovníků$

Počítat se začíná od poslední aktivity v projektu End a přes její předchůdce se pokračuje směrem k aktivitě Start. Následuje ukázkový výpočet pro aktivity uvedené v tabulce 1. Výpočet pro aktivitu End:

- $LF = 22\ (aktivita\ End\ má\ vždy\ LF\ stejné\ jako\ EF)$
- $LS = 22\ (aktivity\ End\ má\ vždy\ LS\ stejné\ jako\ ES)$

Výpočet pro aktivitu H:

- $LF = Miminum\ z\ (End\ LS = 22) \rightarrow 22$
- $LS = 22 - 3 = 19$

Výpočet pro aktivitu G:

- $LF = Minimum\ z\ (H\ LS = 19) \rightarrow 19$
- $LS = 19 - 5 = 14$

Výpočet pro aktivitu E:

- $LF = Minimum\ z\ (H\ LS = 19) \rightarrow 19$
- $LS = 19 - 1 = 18$

Výpočet pro aktivitu F:

- $LF = Minimum\ z\ (F\ LS = 18; G\ LS = 14) \rightarrow 14$
- $LS = 14 - 3 = 11$

Opět se pokračuje stejným způsobem pro všechny aktivity a výpočty se zapíší do tabulky (Tabulka 3), kde je vše daleko přehlednější. Aktivity v projektu se prochází od poslední aktivity End směrem k aktivitě Start.

Posledním krokem je výpočet celkové časové rezervy, což je rozdíl mezi *ES* a *LS* (nebo *EF* a *LF*). U aktivit, které se nacházejí v kritické cestě, se tato veličina rovná 0. Tyto aktivity se nesmí zpozdit, jinak bude prodloužena délka trvání projektu. Celková

časová rezerva je součtem podmíněné časové rezervy a volné časové rezervy. Volná časová rezerva, je taková rezerva, která se může čerpat nezávisle na ostatních aktivitách. Podmíněná časová rezerva naopak udává, že se může čerpat časová rezerva, ale jejím využíváním se užívá volná časová rezerva jiné navazující aktivity.

Tabulka 3 - Výpočet LS, LF a časové rezervy aktivit

Zkratka aktivity	Doba trvání (dny)	Předcházející aktivity	ES	EF	LS	LF	Celková časová rezerva
Start	0	-	0	0	0	0	0
A	3	Start	0	3	0	3	0
B	2	A	3	5	3	5	0
C	6	A, B	5	11	5	11	0
D	2	C	11	13	16	18	5
E	1	D, F	14	15	18	19	4
F	3	C	11	14	11	14	0
G	5	F	14	19	14	19	0
H	3	E, G	19	22	19	22	0
End	0	H	22	22	22	22	0

(zdroj: Autor)

Aktivita E (Tabulka 3) má celkovou časovou rezervu 4. Z vizualizace projektu (Obrázek 6) je patrné, že se jedná o časovou rezervu volnou. Může se čerpat bez ohledu na další aktivity a tím lze posunout začátek aktivity E až do 18. dne. Další aktivita, která má celkovou časovou rezervu rozdílnou od 0 je D (Tabulka 3). Její celková časová rezerva je 5. Po bližším prozkoumání vizualizace projektu (Obrázek 6) je zřejmé, že její celková časová rezerva je složena z volné časové rezervy rovnající se 1 a podmíněné časové rezervy rovnající se 4. To znamená, že lze posunout začátek aktivity D do 17 dne a nemusí se čerpat časová rezervu aktivity jiné. Pokud je však nutné této aktivitě posunout její začátek ještě dále, tak je to možné, ale musí se posunout začátek aktivity E, a tím čerpat její volnou časovou rezervu.

Celý výše popsaný postup v této kapitole značí, jak se postupuje při metodě CPM. V zadání projektu (Tabulka 1) mají aktivity zadanou pouze jednu konkrétní odhadovanou dobu trvání. Takto zadanému projektu se říká deterministický a k výpočtu časové analýzy se využije CPM.

2.3.3 Vypořádání se s nejistým trváním aktivit

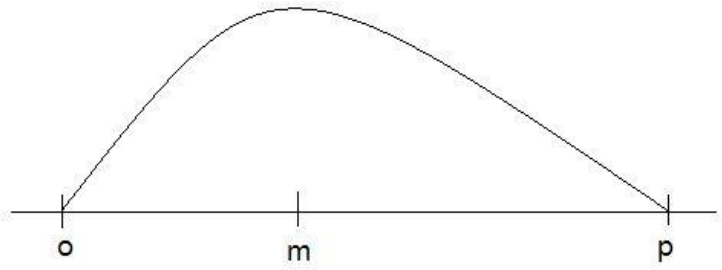
Podle metody CPM (popsané v kapitole Plánování jednotlivých aktivit) vyšlo, že projekt bude trvat 22 dnů. To však předpokládá, že aktuální délka trvání aktivity se bude rovnat odhadnuté době trvání aktivity. V reálném světě je však nemožné odhadnout konkrétní dobu trvání aktivity, jelikož délka trvání každé aktivity je náhodná proměnná z nějakého pravděpodobnostního rozdělení. Tento fakt bere metoda PERT na vědomí. Pokud se v projektu pracuje s aktivitami, jejichž délku trvání reprezentuje nějaké pravděpodobnostní rozdělení, tak se projektu říká stochastický (Hillier a Lieberman, 2001).

V metodě PERT se nejčastěji používá pravděpodobnostní rozdělení Beta k popsání délky trvání jednotlivých aktivit, protože k jeho modelování je potřeba pouze tři údajů o každé aktivitě (Hillier a Lieberman, 2001):

- Nejpravděpodobnější odhad délky trvání aktivity (značka *m*, anglicky most likely estimate)
- Optimistický odhad délky trvání aktivity (zkratka *o*, anglicky optimistic estimate)
- Pesimistický odhad délky trvání aktivity (zkratka *p*, anglicky pessimistic estimate)

Nejčastější vzhled Beta rozdělení je zobrazen na obrázku níže (Obrázek 8), kde osa X znázorňuje uběhnutý čas a osa Y pravděpodobnost. Toto rozdělení je omezeno hodnotami o (optimistický odhad) a p (pesimistický odhad) a je unimodální (má jeden vrchol). Zapisuje se $\beta\langle o; m; p\rangle$. Optimistický i pesimistický odhad by měl být extrémy toho, co je možné a nejpravděpodobnější odhad je nejvyšší bod pravděpodobnostního rozdělení. Občas se může stát, že nejpravděpodobnější odhad se rovná pesimistickému nebo optimistickému odhadu, nebo že se všechny tři odhady rovnají.

Obrázek 8 - Beta rozdělení



(zdroj: Autor)

K práci s Beta rozdělením je třeba znát jeho střední hodnotu, rozptyl a směrodatnou odchylku, které se spočítají takto:

- Zápis statistického rozdělení: $t \sim \beta\langle o; m; p\rangle$
- Střední hodnota: $E(t) = \frac{1}{6} \cdot (o + 4m + p)$
- Rozptyl: $var(t) = \left(\frac{p-o}{4}\right)^2$
- Směrodatná odchylka: $sd(t) = \left(\frac{p-o}{4}\right)$

V tabulce 4 je vidět pozměněné zadání příkladu (Tabulka 1), s kterým se pracovalo, kde místo jednoho odhadnutého času pro každou aktivitu jsou tři odhady, a dále je zde spočtená střední hodnota a rozptyl. V praxi se může stát, že na projekt je určitý časový rámec, který se nemůže překročit. Když se počítalo CPM v předchozí kapitole, tak v zadání (Tabulka 1) projektu se doby trvání aktivit rovnaly sloupečku *m* z tabulky 4. Díky tomu je jasné, že když všechny aktivity budou trvat nejpravděpodobnější odhadnutou dobu, tak projekt bude trvat 22 dnů. Může se však stát, že všechny aktivity budou trvat pesimistický odhad a délka trvání projektu se zvětší.

Tabulka 4 - Pozměněné zadání projektu

Zkratka aktivity	Předchůdci	o	m	p	Střední hodnota	Rozptyl
Start	-	0	0	0	0	0
A	Start	2	3	4	3	0,111
B	A	1	2	3	2	0,111
C	A, B	3	6	7	5,667	0,444

D	C	2	2	4	2,333	0,111
E	D, F	1	1	1	1	0
F	C	2	3	4	3	0,111
G	F	4	5	6	5	0,111
H	E, G	2	3	3	2,833	0,028
End	H	0	0	0	0	0

(zdroj: http://www2.ef.jcu.cz/~jfrieb/tspp/data/pripadove_studie/projekt.pdf)

Stejná cesta v projektu může za určitých předpokladů trvat různé časy. Také se může stát, že se změní i kritická cesta projektu (např. aktivity na původně vypočtené kritické cestě budou ve skutečnosti trvat optimistické doby trvání a aktivity na jiné cestě projektem budou trvat pesimistické doby trvání). Jak tedy lze říct, že projekt bude s určitou pravděpodobností trvat určitou dobu? Existují dva postupy, které předpokládají jisté zjednodušující předpoklady (Hillier a Lieberman, 2001).

Prvním zjednodušujícím předpokladem je, že střední kritická cesta určuje dobu trvání projektu. Ta se zjistí tak, že se postupuje jako u CPM, ale za doby trvání aktivit se berou střední hodnoty jejich beta rozdělení. Z tabulky 5 je vidět, že aktivity, které mají celkovou časovou rezervu 0, leží na střední kritické cestě, takže střední kritická cesta je Start→A→B→C→F→G→H→End. Její střední doba trvání je 21,5 dne (jedná se pouze o hrubý odhad, protože se předpokládá, že všechny aktivity budou mít dobu trvání pohybující se okolo nejpravděpodobnějšího odhadu). Tato střední doba trvání se zjistila jako suma středních hodnot beta rozdělení jednotlivých aktivit, nacházejících se na této cestě. Stejným způsobem se může zjistit střední doba trvání pro každou cestu v projektu.

Tabulka 5 - Výpočet střední kritické cesty

Zkratka aktivity	Předchůdci	Střední hodnota	ES	EF	LS	LF	Celková časová rezerva
Start	-	0	0	0	0	0	0
A	Start	3	0	3	0	3	0
B	A	2	3	5	3	5	0

C	A, B	5,667	5	10,667	5	10,667	0
D	C	2,333	10,667	13	15,334	17,667	4,667
E	D, F	1	13,667	14,667	17,667	18,667	4
F	C	3	10,667	13,667	10,667	13,667	0
G	F	5	13,667	18,667	13,667	18,667	0
H	E, G	2,833	18,667	21,5	18,667	21,5	0
End	H	0	21,5	21,5	21,5	21,5	0

(zdroj: Autor)

Druhým zjednodušujícím předpokladem je, že doby trvání jednotlivých aktivit jsou na sobě nezávislé veličiny. Tento předpoklad se zavádí, aby šlo zjistit rozptyl pro určitou cestu v projektu. Roztyl dvou aktivit se vypočítá podle vztahu:

- $X_A \dots aktivita\ A, X_B \dots aktivita\ B$
- $var(X_A + X_B) = var(X_A) + var(X_B) + 2 \cdot covar(X_A, X_B)$

Když je potřeba zjistit rozptyl dvou aktivit, tak se jedná o součet jejich rozptylů a dvakrát vynásobenou jejich kovarianci. Kovariance představuje vzájemné ovlivnění jednotlivých aktivit. Může k ní nastat, například pokud dvě aktivity vykonává ta samá firma, a když se zpozdí u práce na jedné aktivitě, tak se zvětšuje pravděpodobnost, že se zpozdí i u aktivity druhé. Předpokládá se však, že jednotlivé aktivity jsou na sobě nezávislé a u nezávislých aktivit je kovariance 0. To znamená, že výpočet rozptylu je dán podle vztahu (přesnost výpočtu se snižuje s navzájem závislými aktivitami):

- $X_A \dots aktivita\ A, X_B \dots aktivita\ B, \dots, X_N \dots aktivita\ N$
- $var(X_A + X_B + \cdots + X_N) = var(X_A) + var(X_B) + \cdots + X_N$

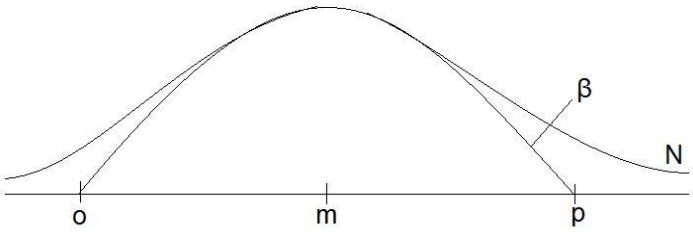
Rozptyl střední kritické cesty (Start→A→B→C→F→G→H→End) je tedy 0,916. U směrodatné odchylky se postupuje takto:

- $X_1 \dots cesta\ v\ projektu$
- $sd(X_1) = \sqrt{var(X_1)}$

Směrodatná odchylka střední kritické cesty (Start→A→B→C→F→G→H→End) se rovná 0,957. Stejným postupem lze zjistit rozptyl a směrodatnou odchylku jakékoliv cesty, která se nachází v projektu.

Třetím a zároveň posledním zjednodušujícím předpokladem je, že platí centrální limitní věta. Tento předpoklad se zavádí, protože je potřeba nějaké pravděpodobnostní rozdělení, které se využije při výpočtech. Centrální limitní věta využívá normálního rozdělení, které je tvarem velmi podobné beta rozdělení (Obrázek 9). Aby pravděpodobnostní odhad byl rozumný, tak by cesta, s kterou se pracuje, měla mít alespoň 21 aktivit. Přesnost se zvyšuje s každou další aktivitou navíc. Všechny pravděpodobnostní výpočty nacházející se v této práci provedené pro normální rozdělení, byly vypočítány pomocí webové aplikace, která je k nalezení na adrese geogebra.org[1].

Obrázek 9 - Porovnání beta a normálního rozdělení



(zdroj: Autor)

Po zavedení těchto předpokladů lze spočítat odpověď na následující otázky, které vystávají při řízení stochastického projektu:

- Otázka 1: Jaká je pravděpodobnost, že délka trvání projektu bude větší nebo rovna než W (žádaná časová hodnota)?
- Otázka 2: Jaká je pravděpodobnost, že délka trvání projektu bude menší nebo rovna než W (žádaná časová hodnota)?
- Otázka 3: Jaká je délka trvání projektu, aby pravděpodobnost, že se stihne, byla Z (žádaná procentní hodnota)?
- Otázka 4: Jaká je délka trvání projektu, aby riziko, že se nestihne, bylo Z (žádaná procentní hodnota)?

První postup, kterým lze odpovědět na výše zmíněné otázky, využívá všech tří předpokladů. Je založen na prvním předpokladu (střední kritická cesta udává délku trvání projektu) a pracuje se pouze s touto cestou (v tomto příkladu je to Start→A→B→C→F→G→H→End). Hodnoty, které se využijí k výpočtům:

- $X_1 \dots cesta$ $\text{Start} \rightarrow \text{A} \rightarrow \text{B} \rightarrow \text{C} \rightarrow \text{F} \rightarrow \text{G} \rightarrow \text{H} \rightarrow \text{End}$
- $T \dots$ $celý\ projekt$

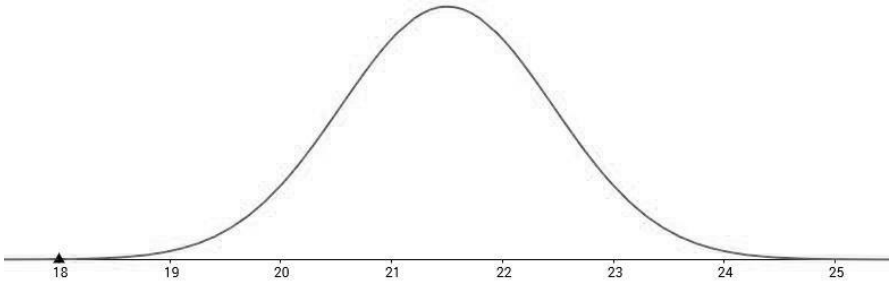
[1] Celý odkaz: https://www.geogebra.org/probability

- $X_1 = T$
- $E(T) = 21{,}5$
- $var(T) = 0{,}916$
- $sd(T) = 0{,}957$

Výpočet a odpověď (Obrázek 10) na otázku 1, pokud W je 18 dnů:

- $T \sim N\langle 21{,}5; 0{,}916\rangle$
- $P(T \leq 18) = 0{,}000128275$
- Pravděpodobnost, že projekt bude trvat 18 dnů nebo méně je přibližně 0,013 %.

Obrázek 10 - Grafická odpověď na otázku 1 podle prvního postupu

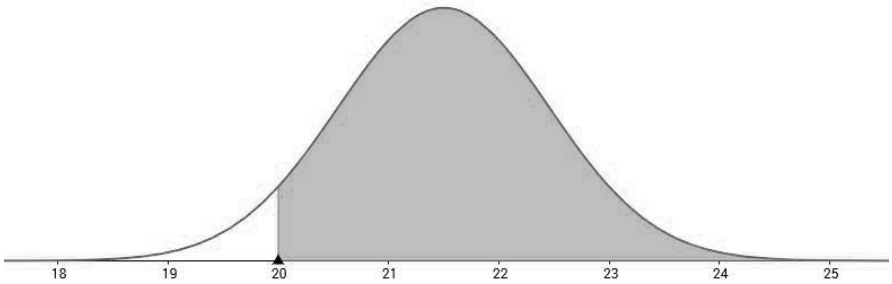


(zdroj: Autor)

Výpočet a odpověď (Obrázek 11) na otázku 2, pokud W je 20 dnů:

- $T \sim N\langle 21{,}5; 0{,}916\rangle$
- $P(T \geq 20) = 1 - P(T \leq 20) = 0{,}941407456$
- Pravděpodobnost, že projekt bude trvat více nebo rovno než 20 dnů je přibližně 94,14 %,

Obrázek 11 - Grafická odpověď na otázku 2 podle prvního postupu

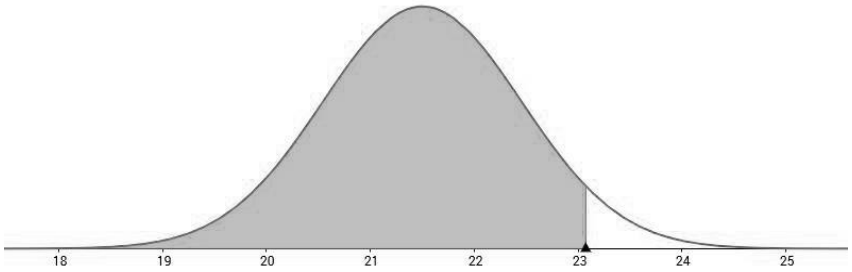


(zdroj: Autor)

Výpočet a odpověď (Obrázek 12) na otázku 3, pokud Z je 95 %:

- $T \sim N\langle 21{,}5;\, 0{,}916\rangle$
- $P(T \leq X) = 0{,}95 \rightarrow \mathrm{X} = 23{,}075$
- Délka trvání projektu je 23,075 dnů, aby pravděpodobnost, že se projekt stihne, byla 95 %.

Obrázek 12 - Grafická odpověď na otázku 3 podle prvního postupu

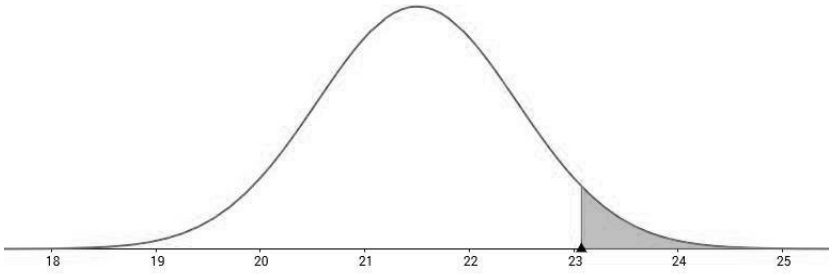


(zdroj: Autor)

Výpočet a odpověď (Obrázek 13) na otázku 4, pokud Z je 10 %:

- $T \sim N\langle 21{,}5;\, 0{,}916\rangle$
- $P(T \geq X) = 0{,}1 \rightarrow \mathrm{P(T \leq X)} = 0{,}9 \rightarrow \mathrm{X} = 22{,}727$
- Délka trvání projektu je 22,727 dnů, aby riziko, že se projekt nestihne, bylo 10 %.

Obrázek 13 - Grafická odpověď na otázku 4 podle prvního postupu



(zdroj: Autor)

Druhý postup, který lze k zodpovězení výše položených otázek využít, odstraňuje první zjednodušující předpoklad. Místo střední kritické cesty se pracuje se všemi cestami v projektu a požadovaný pravděpodobnostní výpočet se počítá pro každou cestu zvlášť. Hodnoty, které jsou třeba k výpočtům:

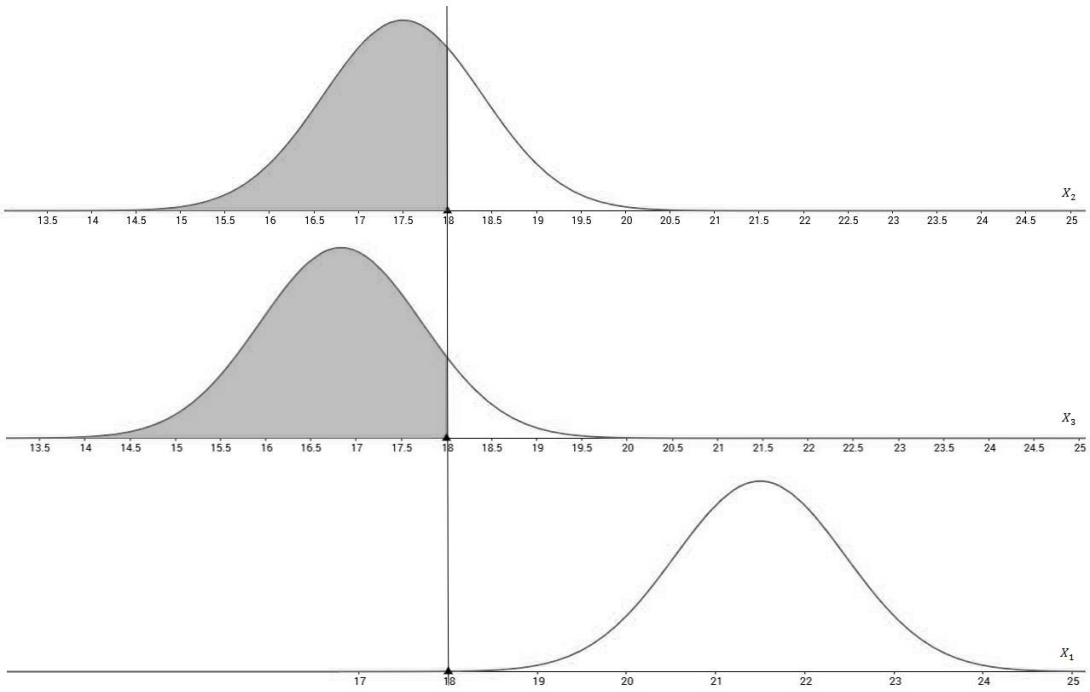
- $X_1 \ldots cesta\ \mathrm{Start \rightarrow A \rightarrow B \rightarrow C \rightarrow F \rightarrow G \rightarrow H \rightarrow End}$
- $X_2 \ldots cesta\ \mathrm{Start \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow H \rightarrow End}$
- $X_3 \ldots cesta\ \mathrm{Start \rightarrow A \rightarrow B \rightarrow C \rightarrow F \rightarrow E \rightarrow H \rightarrow End}$

- $T \ldots celý\ projekt$
- $E(X_1) = 21{,}5$ $E(X_2) = 17{,}5$ $E(X_3) = 16{,}833$
- $var(X_1) = 0{,}916$ $var(X_2) = 0{,}806$ $var(X_3) = 0{,}806$
- $sd(X_1) = 0{,}957$ $sd(X_2) = 0{,}896$ $sd(X_3) = 0{,}896$

Výpočet a odpověď (Obrázek 14) na otázku 1, pokud je W 18 dnů:

- $X_1 \sim N\langle 21{,}5; 0{,}916\rangle;\ X_2 \sim N\langle 17{,}5; 0{,}806\rangle;\ X_3 \sim N\langle 16{,}833; 0{,}806\rangle$
- $P(X_1 \leq 18) = 0{,}000128275$
- $P(X_2 \leq 18) = 0{,}711265669$
- $P(X_3 \leq 18) = 0{,}903176784$
- $P(T \leq 18) = P(X_1 \leq 18) \cap P(X_2 \leq 18) \cap P(X_3 \leq 18) = 0{,}000128275 \cdot 0{,}711265669 \cdot 0{,}903176784 = 0{,}000082403809$
- Pravděpodobnost, že projekt bude trvat 18 dnů nebo méně je přibližně 0,008 %.

Obrázek 14 - Grafická odpověď na otázku 1 podle druhého postupu

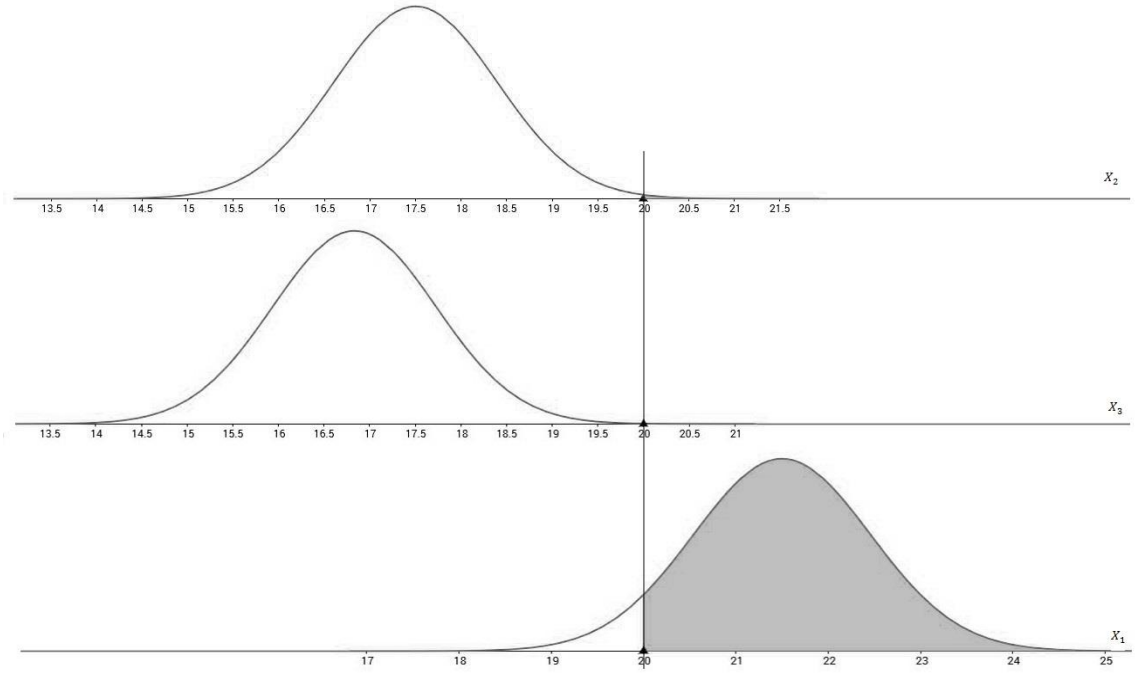


(zdroj: Autor)

Výpočet a odpověď na otázku 2 (Obrázek 15), pokud je W 20 dnů:

- $X_1 \sim N\langle 21{,}5;\ 0{,}916\rangle, X_2 \sim N\langle 17{,}5;\ 0{,}806\rangle,\ X_3 \sim N\langle 16{,}833;\ 0{,}806\rangle$
- $P(X_1 \leq 20) = 0{,}058592544$
- $P(X_2 \leq 20) = 0{,}997327162$
- $P(X_3 \leq 20) = 0{,}999790811$
- $P(T \geq 20) = 1 - [P(X_1 \leq 20) \cap P(X_2 \leq 20) \cap P(X_3 \leq 20)] = 1 - (0{,}058592544 \cdot 0{,}997327162 \cdot 0{,}999790811) = 0{,}941576288954$
- Pravděpodobnost, že projekt bude trvat více nebo rovno než 20 dnů je přibližně 94,15 %.

Obrázek 15 - Grafická odpověď na otázku 2 podle druhého postupu

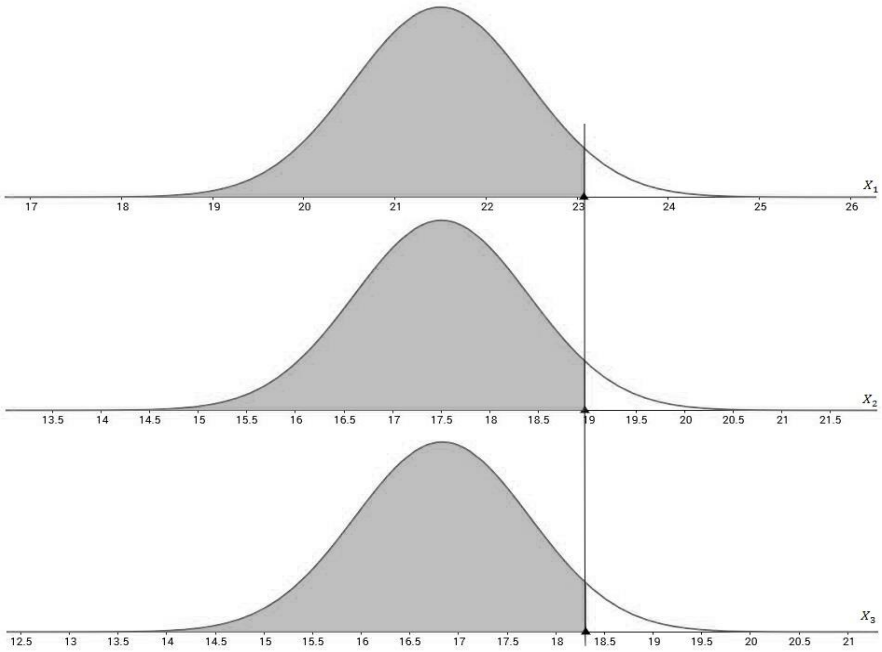


(zdroj: Autor)

Výpočet a odpověď na otázku 3 (Obrázek 16), pokud je *Z* 95 %:

- $X_1 \sim N\langle 21{,}5;\ 0{,}916\rangle, X_2 \sim N\langle 17{,}5;\ 0{,}806\rangle,\ X_3 \sim N\langle 16{,}833;\ 0{,}806\rangle$
- $P(X_1 \leq X) = 0{,}95 \rightarrow \mathrm{X} = 23{,}075$
- $P(X_2 \leq X) = 0{,}95 \rightarrow \mathrm{X} = 18{,}974$
- $P(X_3 \leq X) = 0{,}95 \rightarrow \mathrm{X} = 18{,}307$
- $P(T \leq X) = 0{,}95 \rightarrow \mathrm{X} = \mathrm{Max}[P(X_1 \leq X) = 0{,}95;\ P(X_2 \leq X) = 0{,}95;\ P(X_3 \leq X) = 0{,}95] = Max(23{,}075;\ 18{,}974;\ 18{,}307) = 23{,}075$
- Délka trvání projektu je 23,075 dnů, aby pravděpodobnost, že se projekt stihne, byla 95 %.

Obrázek 16 - Grafická odpověď na otázku 3 podle druhého postupu

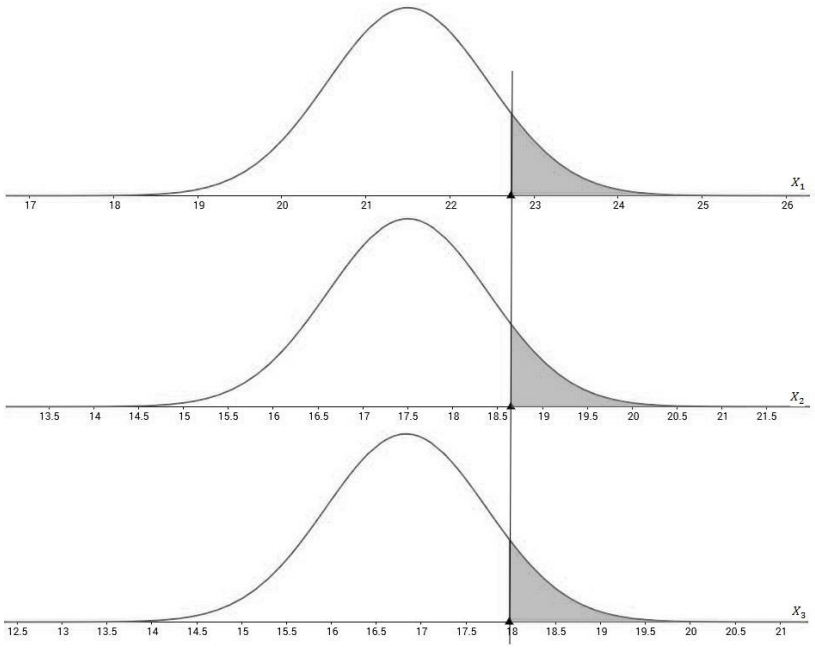


(zdroj: Autor)

Výpočet a odpověď (Obrázek 17) na otázku 4, pokud Z je 10 %:

- $X_1 \sim N\langle 21{,}5;\ 0{,}916\rangle, X_2 \sim N\langle 17{,}5;\ 0{,}806\rangle,\ X_3 \sim N\langle 16{,}833;\ 0{,}806\rangle$
- $P(X_1 \geq X) = 0{,}1 \rightarrow \mathrm{P}(X_1 \leq \mathrm{X}) = 0{,}9 \rightarrow \mathrm{X} = 22{,}727$
- $P(X_2 \geq X) = 0{,}1 \rightarrow \mathrm{P}(X_2 \leq \mathrm{X}) = 0{,}9 \rightarrow \mathrm{X} = 18{,}65$
- $P(X_3 \geq X) = 0{,}1 \rightarrow \mathrm{P}(X_3 \leq \mathrm{X}) = 0{,}9 \rightarrow \mathrm{X} = 17{,}984$
- $P(T \geq X) = 0{,}1 \rightarrow \mathrm{X} = \mathrm{Max}[P(X_1 \leq X) = 0{,}90;\ P(X_2 \leq X) = 0{,}90;\ P(X_3 \leq X) = 0{,}90] = Max(22{,}727;\ 18{,}65;\ 17{,}984) = 22{,}727$
- Délka trvání projektu je 22,727 dnů, aby riziko, že se projekt nestihne, bylo 10 %.

Obrázek 17 - Grafická odpověď na otázku 4 podle druhého postupu



(zdroj: Autor)

2.4 Časově-nákladová analýza

Časově-nákladová analýza se zaměřuje na nalezení správného poměru mezi přímými a nepřímými náklady na projekt. Pod přímými náklady na projekt si lze představit například mzdu dělníků. Tyto náklady v čase klesají (např. zaplatí se dělníkům více, a tudíž oni udělají práci rychleji). Oproti tomu nepřímé náklady v čase rostou. Patří mezi ně například nutná administrativní práce během projektu (čím déle bude projekt trvat, tím více se zaplatí za administrativní práce).

Do tabulky 6 je doplněné zadání z tabulky 1 jehož řešení časové analýzy už je známo z kapitoly Plánování jednotlivých aktivit. Normální náklady představují náklady, které se

na aktivitu vydají, pokud bude trvat normální dobu trvání. Intenzivní doba trvání reprezentuje nejnižší možnou dobu, na kterou lze délku aktivity zkrátit a intenzivní náklady představují náklady vynaložené na aktivitu v případě, že délku trvání aktivity snížíme na intenzivní dobu trvání. Nepřímé náklady na den jsou 2 peněžní jednotky. Marginální cena se vypočítá pro každou aktivitu pomocí vzorce:

- $Marginální\ cena = \frac{Intezivní\ náklady - normální\ nákldy}{Normální\ doba\ trvání - intenzivní\ doba\ trvání}$

Tabulka 6 - Zadání pro časově-nákladovou analýzu

Zkratka aktivity	Doba trvání (dny)	Předcházející aktivity	Normální doba trvání	Intenzivní doba trvání	Normální náklady	Intenzivní náklad	Marginální cena
Start	0	-	0	0	0	0	0
A	3	Start	4	2	1	2	0,5
B	2	A	3	1	2	4	1
C	6	A, B	7	3	15	20	1,25
D	2	C	4	2	10	14	2
E	1	D, F	1	1	5	5	0
F	3	C	4	2	10	14	2
G	5	F	6	4	18	24	3
H	3	E, G	3	2	10	15	5
End	0	H	0	0	0	0	0

(zdroj: http://www2.ef.jcu.cz/~jfrieb/tspp/data/pripadove_studie/projekt.pdf)

Marginální cena představuje cenu, která se zaplatí, když se sníží délka trvání aktivity o den. Tato analýza se snaží zkrátit délku trvání projektu tak, aby se snižovaly nepřímé náklady na projekt, jež rostou s délkou projektu. Jelikož délku projektu určuje kritická cesta, tak pro případné zkracování jsou primární aktivity nacházející se v kritické cestě (Start→A→B→C→F→G→H→End). Snižovat doby trvání aktivit má cenu jen u aktivit, které mají marginální cenu nižší než nepřímé náklady na den. Snaha je zkracovat aktivity, u nichž to vyjde nejlevněji, jsou na kritické cestě a povede to k největšímu časovému zkrácení. Musí se hlídat situace, kdy zkrácením délek trvání aktivit na kritické cestě,

vzniknou kritické cesty jiné. V takovém případě se zkracují jejich společné aktivity, nebo nejlevnější možná kombinace aktivit. Tento výpočet by šel modelovat pomocí lineárního programování. To však patří nad rámec této práce.

Pokud se nezkrátí žádná aktivita, tak je stav nákladů:

- $PN \dots přímé\ náklady, CN \dots celkové\ náklady, NN \dots nepřímé\ náklady$
- $PN = Suma\ všech\ normálních\ nákladů\ aktivit = 71$
- $NN = Suma\ normální\ doby\ trvání\ aktivit\ na\ kritické\ cestě * Nepřímé\ náklady\ na\ den = 27 \cdot 2 = 54$
- $CN = PN + NN = 71 + 54 = 125$

Po zkrácení aktivity A o 1 den:

- $PN \dots přímé\ náklady, CN \dots celkové\ náklady, NN \dots nepřímé\ náklady$
- $PN = Suma\ všech\ normální\ nákladů\ aktivit + náklady\ na\ zkrácení\ dob\ trvání\ aktivit = 71 + 0{,}5 = 71{,}5$
- $NN = Suma\ normální\ doby\ trvání\ aktivit\ na\ kritické\ cestě * Nepřímé\ náklady\ na\ den = 26 \cdot 2 = 52$
- $CN = PN + NN = 71{,}5 + 52 = 123{,}5$

Z výše popsaného výpočtu je vidět, že zkrácením aktivity A o den klesly celkové náklady, protože marginální cena aktivity A je menší než nepřímé náklady na den. Také se snížila doba trvání kritické cesty (a celého projektu) na 26 dnů. Takto je možno aktivitu A zkrátit ještě jednou (více už ne, protože pak se dostane na její intenzivní dobu trvání). Podobně lze hledat další aktivity, které je možné zkrátit. V tomto příkladu to jsou aktivity A, B, C a F, jejichž zkrácení na intenzivní dobu trvání by vedlo k nejnižším možným celkovým nákladům.

2.5 Časově-zdrojová analýza

Časově-zdrojová analýza se zaměřuje na potřebné zdroje k uskutečnění projektu. Konkrétně zjišťuje, jestli využitím časových rezerv jednotlivých aktivit nelze snížit maximální počet zdrojů (př. pracovníků) za časové období. V tabulce 7 je doplněné zadání (Tabulka 1) uvedené výše v této práci, jehož řešení časové analýzy je už známo z kapitoly Plánování jednotlivých aktivit.

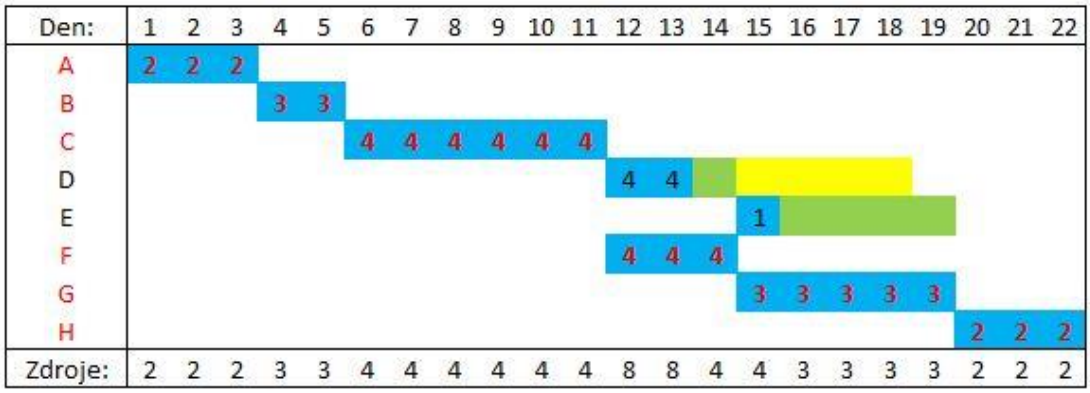
Tabulka 7 - Zadání pro časově-zdrojovou analýzu

Zkratka aktivity	Doba trvání (dny)	Předcházející aktivity	Pracovníci
Start	0	-	0
A	3	Start	2
B	2	A	3
C	6	A, B	4
D	2	C	4
E	1	D, F	1
F	3	C	4
G	5	F	3
H	3	E, G	2
End	0	H	0

(zdroj: http://www2.ef.jcu.cz/~jfrieb/tspp/data/pripadove_studie/projekt.pdf)

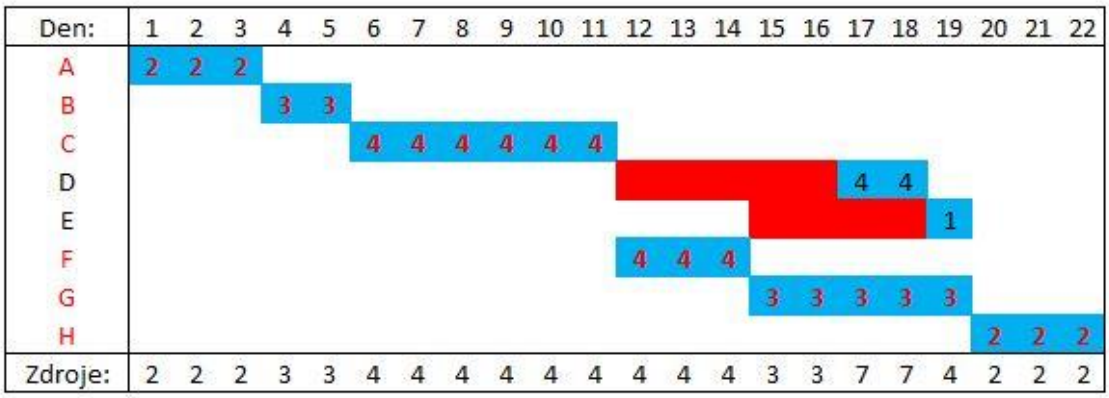
Na obrázku 18 je projekt znázorněný pomocí Ganttovo diagramu, kdy žádná aktivita nevyužívá svoji časovou rezervu. Modrou barvou jsou znázorněné jednotlivé aktivity. Volná časová rezerva je zobrazena zelenou barvou a podmíněná časová rezerva je vyznačena žlutou barvou. V řádku zdroje lze vidět celkový počet pracovníků, kteří jsou ten den potřeba. Z obrázku je vidět, že největší počet pracovníku, kteří jsou potřeba, je 8. Toto číslo lze však snížit, pokud se využije volná časová rezerva aktivity E a podmíněná časová rezerva aktivity D (Obrázek 19). V takovém případě bude stačit pracovníků 7. Tento výpočet by šel modelovat pomocí lineárního programování. To však patří nad rámec této práce.

Obrázek 18 - Ganttův diagram bez využitých časových rezerv



(zdroj: Autor)

Obrázek 19 - Gattův diagram s využitými časovými rezervami



(zdroj: Autor)

2.6 Zhodnocení PERT a CPM

PERT a CPM mají své nedostatky, které mohou být při užití na reálném projektu problém. Jedním z klíčových předpokladů těchto metod je, že aktivita nemůže začít, dokud nejsou dodělány její předchůdci. To však vždy nemusí být pravda a může se stát, že na aktivitě lze začít pracovat, když její předchůdce je z nějaké části už hotový. Tato myšlenka vedla k vytvoření metody PDM (The Precedence Diagramming Method) založené na CPM a PERT, která dovoluje takovýmto způsobem s aktivitami pracovat. Dalším z klíčových předpokladů je, že každá aktivita má vždy k dispozici všechny potřebné zdroje. Mnoho projektů má však omezené zdroje a jejich správná a proveditelná alokace je náročný úkol (jak toho docílit bylo nastíněno v tématu Časově-zdrojová analýza). Přes své nedostatky a zjednodušující předpoklady zůstávají PERT i CPM využívanými metodami pro plánování projektů. Poskytují projektovým manažerům velkou škálu informací, které potřebují (časové rezervy, začátky a konce aktivit, kritickou cestu atd.). Obě

metody vznikly už v 50. letech 20. století a jsou využívané dodnes. To znamená, že prošly řadou testů ze strany reálných projektů, v nichž musely obstát, jinak by se přestaly používat jako některé další metody, které se řízením projektů zabývají. Jejich aplikace na projekt je jednoduchá a při správné modelaci aktivit a zajištění zdrojů výše zmíněné nedostatky nebudou znát (Hillier a Lieberman, 2001).

3 Vývoj aplikace

Aplikace poskytuje uživateli nástroj ke správě projektů a umožňuje vypracování jejich časové analýzy. Uživatel si může vybrat, jestli chce pracovat s deterministickým, nebo stochastickým projektem. Dle jeho výběru musí poskytnout aplikaci potřebná data o délkách trvání jednotlivých aktivit a jejich vzájemnému provázání (nastavení předchůdců a následovníků aktivit). Vstupní data uživatel může načíst i z CSV souboru, který musí mít požadovaný formát. Na základě zadaných dat aplikace vypracuje časovou analýzu projektu, kterou uživateli prezentuje formou třech různých pohledů (síťový graf, Ganttův diagram a tabulkový přehled). Uživatel má k dispozici údaje o všech aktivitách a cestách, které se v projektu nacházejí. Výstupy aplikace je možno exportovat do CSV souboru, s kterým uživatel může provádět své další analýzy. U stochastických projektů je k dispozici uživateli i pravděpodobnostní kalkulátor, který poskytuje nástroj k výpočtu pravděpodobnosti, že se projekt stihne do určitého času apod. K ukládání dat o jednotlivých projektech aplikace využívá svůj vlastní souborový formát NETAN. K instalaci, opravě a odinstalaci aplikace využívá instalační balíčkovací souborový formát MSI, který je určený pro operační systém Windows. Zdrojový kód aplikace je před dekompilací chráněn pomocí obfuskace, kterou provádí aplikace Crypto Obfucator for .NET.

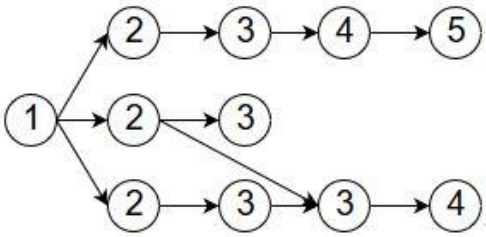
3.1 Metodika

Aplikace je napsána v programovacím jazyku C# pomocí platformy .NET Framework. Tento jazyk a platformu vyvinula a rozvíjí společnost Microsoft, tudíž drtivá většina aplikací vyvinutých pomocí těchto nástrojů jsou primárně určeny pro běh na operačních systémem od této firmy. Konkrétně byl použit .NET Framework 4.5, což znamená, že k zdárné instalaci aplikace je potřeba operační systém Windows Vista, nebo novější operační systém firmy Microsoft a nainstalovaná verze výše zmíněného .NET Frameworku, nebo jeho novější verze. Aplikace byla vyvinuta pomocí IDE (vývojářského prostředí) Visual Studio 2015, které je také produktem firmy Microsoft. Aplikace je určena pro stolní počítače nebo notebooky. Její grafické rozhraní bylo vytvořeno pomocí knihovny WPF (Windows Presentation Foundation) z .NET Frameworku, která využívá značkovací jazyk XAML. K pravděpodobnostním výpočtům využívá open-source knihovnu Math.NET.

3.2 Algoritmus pro prohledávání grafu do šířky

Algoritmus pro prohledávání grafu do šířky (anglicky bradth-first search) patří mezi základní algoritmy pro práci s grafy, který slouží k průchodu grafu. Graf se začne prohledávat z počátečního vrcholu, který se zpracuje. Když se objeví dosud nezpracované potomky počátečního vrcholu, tak se zařadí do fronty a zpracují se stejným způsobem. Tento proces se opakuje, dokud fronta není prázdná. Na obrázku 20 je znázorněný průchod grafem pomocí tohoto algoritmu. Číslo zapsané ve vrcholu znamená, v jaké vlně průchodu grafem byl tento vrchol zpracován (“Algoritmy.net”, 2016).

Obrázek 20 - Algoritmus pro prohledávání grafu do šířky



(zdroj: Autor)

3.3 Návrh modelu tříd

V příloze 1 je vidět návrh diagramu tříd, které náleží logické části aplikace a starají se o výpočet časové analýzy projektu. Všechny třídy z tohoto diagramu jsou popsány v následujících kapitolách. Při návrhu byl kladen důraz na dodržení zásad objektově orientovaného programování. Všechny třídy v diagramu jsou označeny jako Serializable, což znamená, že se mohou binárně serializovat, čehož program využívá k ukládání informací o jednotlivých projektech. Všechny vlastnosti datového typu double ve všech třídách mají ve svém přístupovém objektu set operaci, která do vlastnosti ukládá hodnotu zaokrouhlenou na 8 desetinných míst. Každá třída dědí z třídy NotifyPropertyChangedClass.

3.3.1 Třída NotifyPropertyChangedClass

Abstraktní třída NotifyPropertyChangedClass implementuje rozhraní INotifyPropertyChanged, které se ve WPF využívá k změně grafického rozhraní z kódu. Například když je nějaký popisek z grafického rozhraní propojený s vlastností nějaké třídy a ta vlastnost se změní, tak je žádoucí, aby se změna projevila i v grafickém rozhraní. V takovém případě se musí třídě implementovat rozhraní INotifyPropertyChanged a po změně vlast-

nosti se volá událost PropertyChanged, která přijímá jako první argument objekt, kterému se vlastnost změnila, a jako druhý argument přijímá instanci třídy PropertyChangedEventArgs, jehož konstruktoru se předá název vlastnosti, která se změnila. Pro zjednodušení tohoto procesu třída obsahuje metodu NotifyPropertyChanged (Ukázka kódu 1), která bere za argument název vlastnosti, a sama řeší výše popsanou proceduru. Každá třída, která dědí tuto abstraktní třídu, volá v přístupových objektech set svých vlastností tuto metodu, aby docházelo k aktualizaci grafického rozhraní.

Ukázka kódu 1 - Metoda NotifyPropertyChanged

```
16  public void NotifyPropertyChanged(string propertyName)
17  {
18      if (PropertyChanged != null)
19      {
20          PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
21      }
22  }
```

(zdroj: Autor)

3.3.2 Třída Activity

Po zamyšlení nad problémem časové analýzy projektu je zřejmé, že algoritmy CPM a PERT jsou totožné, jen se liší v délce trvání aktivity, kterou využívají k výpočtu. Tento fakt je vhodné vzít při návrhu tříd v potaz. Třídy představující stochastickou aktivitu (Třída StochasticActivity) a deterministickou aktivitu (Třída DeterministicActivity) obsahují z většiny totožné vlastnosti. Proto obě tyto třídy dědí z abstraktní třídy Activity, která obsahuje všechny společné vlastnosti a metody.

Popis vlastností třídy (v závorkách je uveden datový typ vlastnosti):

- Name (string) – Uchovává informaci o názvu aktivity.
- Description (string) – Uchovává informaci o popisu aktivity.
- IsEnd (bool) – Uchovává informaci, jestli aktivita reprezentuje konec projektu.
- IsStart (bool) – Uchovává informaci, jestli aktivita reprezentuje začátek projektu.
- Successors (List<Activity>) – Kolekce, která obsahuje reference následovníky aktivity.
- Precessors (List<Activity>) – Kolekce, která obsahuje reference na předchůdce aktivity.
- EarliestStart (double) – Uchovává informaci o nejdřívějším možném začátku aktivity.

- EarliestFinish (double) – Uchovává informaci o nejdřívějším možném konci aktivity.
- LatestStart (double) – Uchovává informaci o nejpozdějším možném začátku aktivity.
- LatestFinish (double) – Uchovává informaci o nejpozdějším možném konci aktivity.
- IsCritical (bool) – Uchovává informaci o tom, jestli aktivita leží na kritické cestě.
- TotalTimeReserve (double) – Uchovává informaci o celkové časové rezervě aktivity.
- FreeTimeReserve (double) – Uchovává informaci o volné časové rezervě aktivity.
- DependetTimeReserve (double) – Uchovává informaci o podmíněné časové rezervě aktivity.
- TimeReserveDependetActivities (List<Activity>) – Kolekce, která obsahuje reference na aktivity, které se podílejí na podmíněné časové rezervě aktivity.
- CalculLength (double) – Uchovává informaci o hodnotě, s kterou se má v algoritmech počítat jako s hodnotou představující délku trvání aktivit. Tato vlastnost je označena jako abstract, tudíž si každá dědící třída tuto vlastnost sama implementuje.

Konstruktor třídy je jednoduchý a jen nastaví počáteční hodnoty všem vlastnostem třídy. Podobně funguje i **metoda ClearCalculations**, která slouží k nastavení vlastností představující vypočtené hodnoty do původního stavu.

Metoda CalculateEarliestStartAndFinish (Ukázka kódu 2) spočítá vlastnosti Earliest-Start a EarliestFinish. Správné hodnoty však spočítá jen tehdy, pokud mají všichni její předchůdci již správně spočtené tyto vlastnosti. Vrací hodnotu true, pokud se alespoň jedna z těchto vlastností zvětšila, jinak vrací false. Speciální případ nastane u aktivity, představující začátek projektu. Ta už tyto vlastnosti nastaveny má (jejich hodnoty jsou vždy 0 – počáteční hodnoty), a v tomto případě vrací true.

Ukázka kódu 2 - Metoda CalculateEarliestStartAndFinish

```
143    public bool CalculateEarliestStartAndFinish()
144    {
145        if (IsStart)
146        {
147            return true;
148        }
149
150        double oldEarliestStart = EarliestStart;
151        double oldEarliestFinish = EarliestFinish;
152
153        double newEarliestStart = Precessors.Max(u => u.EarliestFinish);
154        double newEarliestFinish = newEarliestStart + CalculLength;
155
156        if (oldEarliestStart < newEarliestStart || oldEarliestFinish < newEarliestFinish)
157        {
158            EarliestFinish = newEarliestFinish;
159            EarliestStart = newEarliestStart;
160            return true;
161        }
162        return false;
163    }
```

(zdroj: Autor)

Metoda CalculateLatestStartAndFinish (Ukázka kódu 3) určí vlastnosti LatestStart a LatestFinish. Správné hodnoty však vypočítá jen tehdy, pokud mají všichni její následovníci správně spočtené tyto vlastnosti. Vrací hodnotu true, pokud se alespoň jedna z vlastností výpočtem zvětšila, jinak vrací false. Speciální případy jsou aktivity představující začátek a konec projektu. Aktivita představující začátek projektu tyto vlastnosti nastaveny už má (jejich hodnoty jsou vždy 0 – počáteční hodnoty), a aktivitě představující konec projektu se vlastnosti LatestStart a EarliestStart rovnají. To samé u této aktivy platí pro LatestFinish a EarliestFinish. V těchto speciálních případech metoda vždy vrací true.

Ukázka kódu 3 - Metoda CalculateLatestStartAndFinish

```
165     public bool CalculateLatestStartAndFinish()
166     {
167         if (IsEnd)
168         {
169             LatestFinish = EarliestFinish;
170             LatestStart = EarliestStart;
171             return true;
172         }
173         if (IsStart)
174         {
175             return true;
176         }
177
178         double oldLatestFinish = LatestFinish;
179         double oldLatestStart = LatestStart;
180
181         double newLatestFinish = Successors.Min(u => u.LatestStart);
182         double newLatestStart = newLatestFinish - CalculLength;
183
184         if (oldLatestStart < newLatestStart || oldLatestFinish < newLatestFinish)
185         {
186             LatestStart = newLatestStart;
187             LatestFinish = newLatestFinish;
188             return true;
189         }
190         return false;
191     }
```

(zdroj: Autor)

Metoda CalculateOtherTimeAnalysisProperties (Ukázka kódu 4) slouží k výpočtu vlastností TotalTimeReserve, IsCritical, FreeTimeReserve a DependentTimeReserve. Zajímavý je výpočet vlastnosti FreeTimeReserve, která se získá tak, že se projdou všichni následovníci, najde se ten s nejmenší hodnotu uloženou v EarliestStart a od této hodnoty se odečte EarliestFinish aktivity, s kterou se pracuje.

Ukázka kódu 4 - Metoda CalculateOtherTimeAnalysisProperties

```
193    public void CalculateOtherTimeAnalysisProperties()
194    {
195        TotalTimeReserve = LatestStart - EarliestStart;
196        IsCritical = TotalTimeReserve == 0 ? true : false;
197        FreeTimeReserve = Successors.Count == 0 ?
198                    0 : Successors.Min(u => u.EarliestStart) - EarliestFinish;
199        DependentTimeReserve = TotalTimeReserve - FreeTimeReserve;
200    }
201
```

(zdroj: Autor)

Metoda CalculateTimeReserveDependentActivites (Ukázka kódu 5) zjistí aktivity, které se podílejí na podmíněné časové rezervě aktivity, s kterou se pracuje a uloží je do kolekce TimeReserveDependentActivities. Jedná se o pozměněný algoritmus prohledávání grafu do šířky. Procházet se začne graf v aktivitě, která metodu volá. Na řádku 215 se hledají následovníci, kteří nemají vlastnost TotalTimeReserve rovnou 0. Pokud je alespoň jedna takováto aktivita (řádek 217), tak se vstupuje do podmínky. Z vyhledaných následovníků se vybere ten, jehož vlastnost LatestStart je nejmenší (řádek 219). Pokud tato aktivita nemá vlastnost FreeTimeReserve rovnou 0 (řádek 222), tak se přidá do aktivit, které se podílejí na podmíněné časové rezervě (řádek 224). Následně se tato aktivita přidá do fronty, sloužící k procházení grafu (řádek 226).

Ukázka kódu 5 - Metoda CalculateTimeReserveDependentActivites

```
202  public void CalculateTimeReserveDependentActivities()
203  {
204      var timeReserveDependentActivities = new List<Activity>();
205      if (DependentTimeReserve != 0)
206      {
207          var queue = new Queue<Activity>();
208          queue.Enqueue(this);
209
210          while (queue.Count != 0)
211          {
212              var activity = queue.Dequeue();
213              if (activity.Successors.Count != 0)
214              {
215                  var query = activity.successors
216                              .Where(u => u.TotalTimeReserve != 0).Select(u => u);
217                  if (query.Count() != 0)
218                  {
219                      var dependentActivity = query
220                                  .Where(u => u.LatestStart == query.Min(e => e.LatestStart))
221                                  .Select(u => u).First();
222                      if (dependentActivity.FreeTimeReserve != 0)
223                      {
224                          timeReserveDependentActivities.Add(dependentActivity);
225                      }
226                      queue.Enqueue(dependentActivity);
227                  }
228              }
229          }
230      }
231      TimeReserveDependentActivities = timeReserveDependentActivities;
232  }
```

(zdroj: Autor)

3.3.3 Třída Distribution

Další abstraktní třídou je třída Distribution, která představuje pravděpodobnostní rozdělení. Obsahuje tři vlastnosti (v závorkách je uveden datový typ vlastností):

- Median (double) – Uchovává informaci o střední hodnotě.
- Dispersion (double) – Uchovává informaci o rozptylu.

- SnadartDeviation (double) – Uchovává informaci o směrodatné odchylce.

Třída obsahuje dvě metody označené jako abstract. Metoda CalculateDistributionProperties má být volána dědící třídou k nastavení výše popsaných vlastností. Metoda CheckDistributionValues má být volána dědící třídou v momentě, kdy je požadováno zkontrolovat hodnoty zadané pravděpodobnostnímu rozdělení.

3.3.4 Třída Path

Abstraktní třída Path je zde úplně ze stejného důvodu, jako třída Activity. Uchovává informace o cestě projektem, dědí z ní třída představující deterministickou cestu (Třída DeterministicPath) a třída představující stochastickou cestu (Třída StochasticPath). Také však implementuje rozhraní ICloneable, které je využito v metodě Clone. Popis vlastností (v závorkách je uveden datový typ):

- Activities (List<Activity>) – Kolekce obsahuje reference na aktivity v cestě.
- IsEnded (bool) – Vlastnost (Ukázka kódu 6), říkající informaci, jestli je cesta ukončena. Pokud poslední aktivita v Activities, představuje konec projektu, tak vrací true, jinak vrací false.

Ukázka kódu 6 - Vlastnost IsEnded

```
20    public bool IsEnded { get { return Activities == null || Activities.Count == 0 ?
21                        false : Activities[Activities.Count - 1].IsEnd; } }
```

(zdroj: Autor)

- Name (string) – Vlastnost uchovává informaci o názvu cesty.
- Length (double) – Vlastnost uchovává informaci o délce cesty.
- IsCritical (bool) – Vlastnost uchovává informaci o tom, jestli je cesta kritická.
- TotalTimeReserve (double) – Vlastnost uchovává informaci to celkové časové rezervě aktivit v cestě.
- TotalTimeReserveActivites (List<Activity>) – Kolekce obsahující reference na aktivity, které se podílejí na celkové časové rezervě cesty.

Konstruktor třídy je jednoduchý a pouze nastavuje počáteční hodnoty vlastnostem.

Metoda CalculateTimeAnalysisProperties (Ukázka kódu 7) nastaví hodnoty vlastnostem Length, IsCritical, TotalTimeReserve a TotalTimeReserveActivities. Vlastnost Length je vypočtena způsobem, který byl popsán v kapitole Kritická cesta. Vlastnost IsCritical se nastaví tak, že je nalezena maximální hodnota vlastnosti TotalTimeReserve v kolekci aktivit Activities (řádek 87). Pokud je tato hodnota 0, tak je IsCritical true, jinak se nastaví jako false. TotalTimeReserve je nastavena jako suma hodnot vlastnosti FreeTimeReserve patřící aktivitám v kolekci Activities (řádek 88). TotalTimeReserveActivites se nastaví jako výběr z kolekce Activities, kde se vyberou aktivity, které nemají vlastnost FreeTimeReserve rovnou 0 (řádek 89).

Ukázka kódu 7 – Metoda CalculateTimeAnalysisProperties

```
74  public void CalculateTimeAnalysisProperties()
75  {
76      double length = 0;
77      foreach (var activity in Activities)
78      {
79          if (activity.IsEnd)
80          {
81              continue;
82          }
83          length += activity.EarliestStart == length
84                      ? activity.CalculLength : activity.EarliestStart - length + activity.CalculLength;
85      }
86      Length = length;
87      IsCritical = Activities.Max(u => u.TotalTimeReserve) == 0 ? true : false;
88      TotalTimeReserve = Activities.Sum(u => u.FreeTimeReserve);
89      TotalTimeReserveActivities = Activities.Where(u => u.FreeTimeReserve != 0).Select(u => u).ToList();
90  }
```

(zdroj: Autor)

Metoda GetSumOfActivitiesHashCodes (Ukázka kódu 8) vrací sumu hodnot, které se získají tak, že se projdou všechny aktivity v kolekci Activites a zavolá se jejich metoda GetHashCode. Tato metoda vrací unikátní číslo, pro každou vytvořenou instanci třídy, a mohou se podle ní objekty porovnávat. Když zavolají tuto metodu dvě instance třídy Path, které budou mít v kolekci Activities reference na stejné objekty, tak tato metoda vrátí stejnou hodnotu.

Ukázka kódu 8 - Metoda GetSumOfActivitesHashCodes

```
92  public int GetSumOfActivitiesHashCodes()
93  {
94      int hashCode = 0;
95      foreach (var activity in Activities)
96      {
97          hashCode += activity.GetHashCode();
98      }
99      return hashCode;
100 }
```

(zdroj: Autor)

Metoda Clone (Ukázka kódu 9) slouží k naklonování cesty. Jelikož je potřeba určitá hloubka kopie, tak vzniklému klonu, který je získán metodou MemberwiseClone, jsou vytvořeny nové kolekce pro vlastnosti Activites a TotalTimeReserveActivites, do kterých jsou zkopírovány reference na objekty kopírované instance.

Ukázka kódu 9 - Metoda Clone

```
102     public object Clone()
103     {
104         var clone = this.MemberwiseClone();
105         (clone as Path).Activities = new List<Activity>();
106         (clone as Path).TotalTimeReserveActivities = new List<Activity>();
107         (clone as Path).Activities.AddRange(Activities);
108         return clone;
109     }
```

(zdroj: Autor)

3.3.5 Třída DeterministicActivity

Třída DeterministicActivity dědí z třídy Activity a reprezentuje deterministickou aktivitu. Obsahuje vlastnost Length, která je datového typu double a uchovává informaci o délce trvání aktivity. Přepisuje abstraktní vlastnost CalculLength rodičovské třídy tak, aby tato vlastnost vracela hodnotu vlastnosti Length. Konstruktor třídy nastavuje vlastnost Length na počáteční hodnotu 0 a volá konstruktor rodičovské třídy.

3.3.6 Třída DeterministicPath

Třída DeterministicPath dědí z třídy Path a představuje deterministickou cestu projektem. Neobsahuje žádné další metody a vlastnosti, protože všechno potřebné už obsahuje rodičovská třída.

3.3.7 Třída BetaDistribution

Třída BetaDistribution dědí z třídy Distribution. Reprezentuje Beta pravděpodobnostní rozdělení. Popis vlastností (v závorkách je uveden datový typ):

- ShortestLength (double) – Uchovává informaci o optimistickém odhadu.
- LongestLength (double) – Uchovává informaci o pesimistickém odhadu.
- MostLikelyLength (double) – Uchovává informaci o nejpravděpodobnějším odhadu.

Konstruktor třídy nastavuje výše popsané vlastnosti na počáteční hodnoty.

Přepsaná metoda rodičovské třídy CalculateDistributionProperties (Ukázka kódu 10) počítá vlastnosti Median, StandartDeviation a Dispersion.

Ukázka kódu 10 - Metoda CalculateDistributionProperties

```
48    public override void CalculateDistributionProperties()
49    {
50        Median = (ShortestLength + 4 * MostLikelyLength + LongestLength) / 6;
51        StandartDeviation = (LongestLength - ShortestLength) / 6;
52        Dispersion = Math.Pow(StandartDeviation, 2);
53    }
54
```

(zdroj: Autor)

Přepsaná metoda CheckDistributionValues (Ukázka kódu 11) kontroluje hodnoty zadané ve vlastnostech třídy. ShortestLength musí být menší nebo roven než LongestLength, a MostLikelyLength musí být v uzavřeném intervalu tvořícím hodnotami ShortestLength a LongestLength. Pokud tyto podmínky neplatí, je vyvolána výjimka.

Ukázka kódu 11 - Metoda CheckDistributionValues

```
55    public override void CheckDistributionValues()
56    {
57        if (LongestLength < ShortestLength)
58        {
59            throw new Exception("Beta rozdělení má zadanou horní mez menší než dolní mez.");
60        }
61        else if (MostLikelyLength > LongestLength || MostLikelyLength < ShortestLength)
62        {
63            throw new Exception("Beta rozdělení má zadanou nejpravděpodobnější hodnotu,
64                + "která se nenachází v zadaných mezích.");
65        }
66    }
```

(zdroj: Autor)

3.3.8 Třída StochasticActivity

Třída StochasticActivity dědí z třídy Activity a reprezentuje stochastickou aktivitu. Obsahuje vlastnost Length, která je datového typu třídy Distribution. Přepisuje abstraktní vlastnost CalculLength (Ukázka kódu 12) rodičovské třídy tak, aby tato vlastnost vracela hodnotu vlastnosti Median vlastnosti Length. Konstruktor ukládá do vlastnosti Length novou instanci objektu BetaDistribution a volá konstruktor rodičovské třídy.

Ukázka kódu 12 - Vlastnost CalculLength

```
21    public override double CalculLength { get { return Length.Median; } }
```

(zdroj: Autor)

3.3.9 Třída StochasticPath

Třída SchosticPath reprezentuje stochastickou cestu projektem a dědí ze třídy Path. Oproti třídě Path má navíc níže uvedené vlastnosti (v závorkách je uveden datový typ):

- Median (double) – Uchovává informaci o střední hodnotě délky trvání cesty.
- Dispersion (double) – Uchovává informaci o rozptylu cesty.
- StandartDeviation (double) – Uchovává informaci o směrodatné odchylce cesty.

Metoda CalculateStatisticValues (Ukázka kódu 13) počítá a ukládá hodnoty do vlastností Median, Dispersion a StandartDeviation. Vlastnost Median se vypočítá jako suma středních délek trvání jednotlivých aktivit v cestě (řádek 40). Do vlastnosti Dispersion se dosadí suma rozptylů aktivit v cestě (řádek 41) a StandartDeviation se rovná odmocnině z rozptylu (řádek 42).

Ukázka kódu 13 - Metoda CalculateStatisticValues

```
38  public void CalculateStatisticValues()
39  {
40      Median = Activities.Sum(u => (u as StochasticActivity).Length.Median);
41      Dispersion = Activities.Sum(u => (u as StochasticActivity).Length.Dispersion);
42      StandartDeviation = Math.Sqrt(Dispersion);
43  }
```

(zdroj: Autor)

Metoda NormalDistribution (Ukázka kódu 14) si bere za argument časovou jednotku a vrací pravděpodobnost, že cesta bude trvat méně nebo rovno než předaná časová jednotka. K výpočtu je využita třída Normal z knihovny Math.NET, která představuje normální rozdělení.

Ukázka kódu 14 - Metoda NormalDistribution

```
84  public double NormalDistribution(double timeUnit)
85  {
86      return new Normal(Median, StandartDeviation).CumulativeDistribution(timeUnit);
87  }
```

(zdroj: Autor)

Metoda InversionNormalDistribution (Ukázka kódu 15) si bere za argument pravděpodobnostní jednotku a vrací časovou jednotku, v které normální rozdělení cesty dosahuje zadané pravděpodobnosti. K výpočtu je využita opět třída Normal z knihovny Math.NET.

Ukázka kódu 15 - Metoda InversionNormalDistribution

```
89  public double InversionNormalDistribution(double probability)
90  {
91      return new Normal(Median, StandartDeviation).InverseCumulativeDistribution(probability);
92  }
```

(zdroj: Autor)

3.3.10 Třída Project

Třída Project obaluje třídu Solution a stará se o ukládání dat o projektu. Vlastnost Solution datového typu Solution (třída Solution je vysvětlena dále v této práci) slouží k uchování vypočítaných informací o projektu. Vlastnost FileName datového typu string uchovává celý název souboru (včetně cesty k němu), do kterého se mají informace o projektu ukládat. Vlastnost Name (Ukázka kódu 16) datového typu string má pouze přístupový objekt get a vrací jen název souboru, od kterého odtrhává koncovku souborového typu NETAN.

Ukázka kódu 16 - Vlastnost Name

```
32  public string Name
33  {
34      get
35      {
36          if (string.IsNullOrWhiteSpace(FileName) || !FileName.Contains("\\"))
37          {
38              return string.Empty;
39          }
40          return FileName.Substring(FileName.LastIndexOf("\\") + 1).Replace(".netan", "");
41      }
42  }
```

(zdroj: Autor)

Konstruktor třídy (Ukázka kódu 17) si bere jako argumenty výčtový typ enum SolutionType (Ukázka kódu 18), který předává dále konstruktoru třídy Solution a instanci této třídy ukládá do vlastnosti Solution (řádek 45). Dále si jako argument bere název souboru, do kterého se mají ukládat informace o projektu a tuto hodnotu ukládá do vlastnosti FileName (řádek 46).

Ukázka kódu 17 - Konstruktor třídy Project

```
44  public Project(SolutionType solutionType = SolutionType.Deterministic, string fileName = null)
45  {
46      Solution = new Solution(solutionType);
47      FileName = string.IsNullOrWhiteSpace(fileName) ? GetFreeFileName() : fileName;
48  }
```

(zdroj: Autor)

Ukázka kódu 18 - Výčtový typ SolutionType

```
15  public enum SolutionType
16  {
17      Deterministic, Stochastic
18  }
```

(zdroj: Autor)

Metoda Save (Ukázka kódu 19) ukládá informace o projektu. K ukládání dat je využita binární serializace. Tento způsob ukládání dat je jednoduchý na implementaci a zvládá všechny vlastnosti, které jsou požadovány (uchování referencí na instance objektů v jednotlivých kolekcích tříd, možnost uložit si data bez připojení k internetu). V podstatě se při tomto způsobu ukládání dat objekt, s kterým pracuje aplikace, přemění na proud bytů a tento proud se uloží do nějakého souboru. Nejdříve se zkontroluje, jestli soubor se jménem, do kterého se mají informace uložit, již existuje (řádek 68). Pokud existuje, tak je odstraněn (řádek 7,). Dále z vlastnosti FileName je zjištěn název adresáře, v kterém se pracuje (řádek 72). V tomto adresáři se následně vytvoří dočasný adresář (řádek 73 a 75). Do tohoto dočasného adresáře se vytvoří soubor File.bin a pomocí binárního serializace jsou uložena do tohoto souboru data o instanci objektu, s kterým se pracuje (řádek 77 až 82). Následně je tento adresář zabalen pomocí kompresního formátu ZIP a tento zabalený soubor je pojmenován podle vlastnosti FileName (řádek 84). Dočasný adresář se nakonec smaže i se vším, co obsahuje (řádek 86).

Ukázka kódu 19 - Metoda Save

```
66  public void Save()
67  {
68      if (File.Exists(FileName))
69      {
70          File.Delete(FileName);
71      }
72      string fileDirectory = FileName.Remove(FileName.LastIndexOf("\\"));
73      string tempDirectory = Path.Combine(fileDirectory, Path.GetRandomFileName());
74
75      Directory.CreateDirectory(tempDirectory);
76
77      using (var fileStream = File.Create(Path.Combine(tempDirectory, "File.bin")))
78      {
79          var binaryFormatter = new BinaryFormatter();
80          binaryFormatter.Serialize(fileStream, this);
81          fileStream.Close();
82      }
83
84      ZipFile.CreateFromDirectory(tempDirectory, FileName);
85
86      Directory.Delete(tempDirectory, true);
87  }
```

(zdroj: Autor)

Statická metoda Load (Ukázka kódu 20) si bere za argument název souboru, z kterého se mají načíst data o projektu. Nejdříve je načten proud bytů tohoto souboru (řádek 93). Tento proud je následně předán konstruktoru třídy ZipArchive, která umožňuje pracovat se soubory komprimovány kompresním formátem ZIP. Následně je v tomto komprimovaném archivu nalezen soubor File.bin (řádek 97) a otevřen jeho proud bytů (řádek 98). Tento proud je předán instanci třídy BinaryFormatter, který ho deserializuje a vrátí objekt datového typu Project (řádek 100 až 102).

Ukázka kódu 20 - Metoda Load

```
89  public static Project Load(string fileName)
90  {
91      Project project;
92
93      using (var fileStream = File.OpenRead(fileName))
94      {
95          using (var zipArchive = new ZipArchive(fileStream))
96          {
97              ZipArchiveEntry binaryFile = zipArchive.GetEntry("File.bin");
98              using (var stream = binaryFile.Open())
99              {
100                 var binaryFormatter = new BinaryFormatter();
101                 project = binaryFormatter.Deserialize(stream) as Project;
102                 stream.Close();
103             }
104             zipArchive.Dispose();
105         }
106         fileStream.Close();
107     }
108
109     return project;
110 }
```

(zdroj: Autor)

3.3.11 Třída Solution

Třída Solution je nejdůležitější částí celé logické části aplikace. Obsahuje všechny informace o projektu, volá výpočetní metody, obsluhuje metody jednotlivých aktivit a cest a volá je ve správné posloupnosti, aby výpočty byly provedeny správně. Má vlastnosti (v závorkách je uveden datový typ vlastnosti):

- TimeAnalysisCalculated (bool) – Uchovává informaci o tom, jestli byla vypočítána časová analýza projektu.
- Activities (List<Activity>) – Kolekce uchovávající reference na aktivity v projektu.
- Paths (List<Path>) – Kolekce uchovávající reference na cesty v projektu.
- SolutionType (SolutionType) – Uchovává informaci o druhu projektu.

Konstruktor třídy (Ukázka kódu 21) si bere za argument druh projektu a tuto hodnotu uloží do vlastnosti SolutionType. Dále nastaví vlastnostem počáteční hodnoty a do vlastnosti Activities uloží pomocné aktivity představující začátek a konec projektu.

Ukázka kódu 21 - Konstruktor třídy Solution

```
66  public Solution(SolutionType solutionType)
67  {
68      SolutionType = solutionType;
69      Activities = new List<Activity>();
70      Paths = new List<Path>();
71      TimeAnalysisCalculated = false;
72
73      if (SolutionType == SolutionType.Deterministic)
74      {
75          Activities.Add(new DeterministicActivity() { Name = "Start", IsStart = true,
76              Description = "Na tuto aktivitu navažte startovní aktivity(u) projektu." });
77          Activities.Add(new DeterministicActivity() { Name = "End", IsEnd = true,
78              Description = "Na tuto aktivitu navažte konečné(ou) aktivity(u) projektu." });
79      }
80      else
81      {
82          Activities.Add(new StochasticActivity() { Name = "Start", IsStart = true,
83              Description = "Na tuto aktivitu navažte startovní aktivity(u) projektu." });
84          Activities.Add(new StochasticActivity() { Name = "End", IsEnd = true,
85              Description = "Na tuto aktivitu navažte konečné(ou) aktivity(u) projektu." });
86      }
87  }
```

(zdroj: Autor)

Metoda CalculateSolution (Ukázka kódu 22) slouží k řešení analýzy projektu. Nejdříve volá metodu ClearCalculations, která vymaže již spočtené řešení analýz. Poté je volána metoda GraphConnectivityValidation, která kontroluje, jestli jsou aktivity v projektu správně provázány. Pokud je projekt stochastický (řádek 93), tak je zavolána metoda DistributionValidation kontrolující hodnoty zadané pravděpodobnostnímu rozdělení aktivit. Dále je zavolána metoda CalculateTimeAnalysis, která vypočítá časovou analýzu projektu. Pak jsou z vlastnosti Path vymazány tranzitivní cesty pomocí metody RemoveTransitivePaths. Následně jsou aktivity uspořádány do vlastnosti Activites podle vlastností EarlistStart, CalculLength a Name. Cesty jsou do vlastnosti Paths seřazeny podle vlastností IsCritical, Length, a Activites.Count.

Ukázka kódu 22 - Metoda CalculateSolution

```
89  public void CalculateSolution()
90  {
91      ClearCalculations();
92      GraphConnectivityValidation();
93      if (SolutionType == SolutionType.Stochastic)
94      {
95          DistributionValidation();
96          CalculateDistributionValues();
97      }
98      FindAllPaths();
99
100     CalculateTimeAnalysis();
101     RemoveTransitivePaths();
102
103     Activities = Activities.OrderBy(u => u.EarliestStart)
104         .ThenBy(u => u.CalculLength).ThenBy(u => u.Name).ToList();
105     Paths = Paths.OrderByDescending(u => u.IsCritical)
106         .ThenByDescending(u => u.Length)
107         .ThenByDescending(u => u.Activities.Count).ToList();
108 }
```

(zdroj: Autor)

Metoda ClearCalculations (Ukázka kódu 23), nastaví vlastnost TimeAnalysisCalculated na hodnotu false, vymaže z vlastnosti Paths všechny reference na objekty, a pak proiteruje všechny aktivity ve vlastnosti Activities a zavolá jejich metodu ClearCalculations.

Ukázka kódu 23 - Metoda ClearCalculations

```
110     public void ClearCalculations()
111     {
112         TimeAnalysisCalculated = false;
113         TimeResourcesAnalysisCalculated = false;
114         TimeCostsAnalysisCalculated = false;
115         Paths.Clear();
116         foreach (var activity in Activities)
117         {
118             activity.ClearCalculations();
119         }
120     }
```

(zdroj: Autor)

Metoda GraphConnectivityValidation (Ukázka kódu 24), slouží k ověření, zda je graf souvislý. Ve vlastnosti Activites je nalezena aktivita představující začátek projektu (řádek 124). Následně se vytvoří kolekce s aktivitami, které byly již zpracované a je přidána do této kolekce startovní aktivita (řádek 126 a 127). Pak se vytvoří fronta, která je využita k prohledání grafu do šířky (řádek 129 a 130), do které je přidána startovní aktivita. Dokud není fronta prázdná, tak probíhá cyklus, v kterém je vyřazen prvek z fronty (řádek 134), jsou proiterováni všichni jeho následovníci (řádek 135), a pokud navštívené aktivity neobsahují následovníka aktivity (řádek 137), tak je tento následovník přidán do fronty a navštívených aktivit. Když se počet navštívených aktivit nerovná počtu aktivit ve vlastnosti Activities (řádek 145), tak je vyvolána výjimka, která obsahuje zprávu o aktivitách, které nejsou spojené s počáteční aktivitou žádnou cestou (řádek 147 až 150).

Ukázka kódu 24 - Metoda GraphConnectivityValidation

```
122  private void GraphConnectivityValidation()
123  {
124      var startingActivity = Activities.First(u => u.IsStart);
125
126      var visitedActivities = new List<Activity>();
127      visitedActivities.Add(startingActivity);
128
129      var queue = new Queue<Activity>();
130      queue.Enqueue(startingActivity);
131
132      while (queue.Count != 0)
133      {
134          var activity = queue.Dequeue();
135          foreach (var successor in activity.Successors)
136          {
137              if (!visitedActivities.Contains(successor))
138              {
139                  visitedActivities.Add(successor);
140                  queue.Enqueue(successor);
141              }
142          }
143      }
144
145      if (visitedActivities.Count != Activities.Count)
146      {
147          throw new Exception("Na následující aktivity se pomocí zadaných propojení " +
148              "nedá dostat z kořenové aktivity \"Start\":"
149              + Environment.NewLine
150              + string.Join("-", Activities.Except(visitedActivities).Select(u => u.Name)));
151      }
152  }
```

(zdroj: Autor)

Metoda DistributionValidation (Ukázka kódu 25) slouží k ověření hodnot zadaných pravděpodobnostnímu rozdělení. Na počátku metody je vytvořena proměnná, do které se bude ukládat zpráva pro výjimku (řádek 156). Poté se projdou všechny aktivity v projektu (řádek 157) a v bloku na odchytávání výjimek je zavolána metodu CheckDistributionValues, která vyvolá výjimku, pokud jsou zadány nevalidní hodnoty (řádek 161). Pokud tato metoda vyvolá výjimku, tak je její zpráva uložena do proměnné vytvořené na začátku metody (řádek 165). Následně se zkontroluje, jestli v proměnné je uložena nějaká zpráva (řádek 169). Pokud ano, tak je vyvolána výjimka se zprávou (řádek 170).

Ukázka kódu 25 - Metoda DistributionValidation

```
154     private void DistributionValidation()
155     {
156         string message = string.Empty;
157         foreach (var activity in Activities)
158         {
159             try
160             {
161                 (activity as StochasticActivity).Length.CheckDistributionValues();
162             }
163             catch (Exception ex)
164             {
165                 message += string.Format("{0} - {1}{2}",
166                     activity.Name, ex.Message, Environment.NewLine);
167             }
168         }
169         if (!string.IsNullOrWhiteSpace(message))
170         {
171             throw new Exception(message);
172         }
173     }
```

(zdroj: Autor)

Metoda FindAllPaths (Ukázka kódu 26) slouží k nalezení všech cest v projektu. Nejprve je nalezena aktivita představující počátek projektu (řádek 185). Následně se vytvoří fronta, do které je tato aktivita přidána (řádek 189 a 188). Na řádku 190 je vytvořena nová kolekce na ukládání nalezených cest v projektu. V závislosti na druhu projektu je vytvořena první cesta projektem, do jejichž aktivit se přidá počáteční aktivita (řádek 191 až 201). Tato cesta je následně přidána do nalezených cest (řádek 202). Následuje upravený kód pro prohledávání grafu do šířky. Z fronty je vyndána aktivita (řádek 206). Následně jsou nalezeny v proměnné obsahující všechny cesty ty, které končí vyndanou aktivitou z fronty (řádek 207). Všechny takto nalezené cesty se naklonují a přidají do nalezených cest (řádek 209 až 215). Poté jsou proiterováni všichni následovníci hlavní aktivity, se kterou se pracuje (řádek 216), naleznou se všechny cesty, které končí hlavní aktivitou cyklu a mají různou hodnotu, kterou vrací metoda GetSumOfActivitesHashCodes (řádek 218). Všechny takto nalezené cesty jsou proiterovány (řádek 223). Nejdříve je zkontrolováno zacyklení cesty pomocí metody CycleValidation, které je předána cesta a následovník hlavní aktivity (řádek 225), následně je tato cesta rozšířena o tohoto následovníka (řádek 226). Pokud takto nalezených cest nebylo 0 (228), tak je následovník přidán do fronty užité pro procházení grafu (řádek 230).

Ukázka kódu 26 - Metoda FindAllPaths

```
183     private void FindAllPaths()
184     {
185         var startingActivity = Activities.First(u => u.IsStart);
186 
187         var queue = new Queue<Activity>();
188         queue.Enqueue(startingActivity);
189 
190         var paths = new List<Path>();
191         Path firstPath = null;
192         if (SolutionType == SolutionType.Deterministic)
193         {
194             firstPath = new DeterministicPath();
195         }
196         else
197         {
198             firstPath = new StochasticPath();
199         }
200 
201         firstPath.Activities.Add(startingActivity);
202         paths.Add(firstPath);
203 
204         while (queue.Count != 0)
205         {
206             var activity = queue.Dequeue();
207             var pathsToClone = paths.FindAll(u => (u.Activities.Count == 0
208             ? null : u.Activities.Last()) == activity && !u.IsEnded).ToList();
209             foreach (var pathToClone in pathsToClone)
210             {
211                 for (int i = 0; i < activity.Successors.Count - 1; i++)
212                 {
213                     paths.Add(pathToClone.Clone() as Path);
214                 }
215             }
216             foreach (var successor in activity.Successors)
217             {
218                 var pathsToAjdust = (from p in paths
219                                      where (p.Activities.Count == 0
220                                      ? null : p.Activities.Last()) == activity && !p.IsEnded
221                                      group p by p.GetSumOfActivitiesHashCodes() into myGroup
222                                      select myGroup.First()).ToList();
223                 foreach (var pathToAdjust in pathsToAjdust)
224                 {
225                     CycleValidation(pathToAdjust, successor);
226                     pathToAdjust.Activities.Add(successor);
227                 }
228                 if (pathsToAjdust.Count != 0)
229                 {
230                     queue.Enqueue(successor);
231                 }
232             }
233         }
234 
235         Paths.AddRange(paths);
236     }
```

(zdroj: Autor)

Metoda CycleValidation (Ukázka kódu 27) vyvolá výjimku, pokud metodě argumentem předaná cesta již obsahuje metodě argumentem předanou aktivitu (řádek 264). Následně sestaví zprávu vypisující aktivity, na kterých bylo zacyklení nalezeno a tuto zprávu předá nové instanci třídy Exception.

Ukázka kódu 27 - Metoda CycleValidation

```
262  private void CycleValidation(Path pathToAdjust, Activity activityToAdd)
263  {
264      if (pathToAdjust.Activities.Contains(activityToAdd))
265      {
266          string message = "Nalezeno zacyklení na aktivitách:" + Environment.NewLine;
267          string path = "";
268          bool write = false;
269          foreach (var activity in pathToAdjust.Activities)
270          {
271              if (activity.Name == activityToAdd.Name)
272              {
273                  write = !write;
274              }
275              if (write)
276              {
277                  path += activity.Name + "-";
278              }
279          }
280          path += activityToAdd.Name;
281          message += path;
282          throw new Exception(message);
283      }
284  }
```

(zdroj: Autor)

Metoda CalculateTimeAnalysis (Ukázka kódu 28) slouží k výpočtu časové analýzy projektu. Nejdříve se zavolají metody SetEarliestStartAndFinishProperties a SetLatestStartAndFinishProperties (řádky 288 a 289), které aktivitám v projektu nastaví vlastnosti EarliestStart, EarlistFinish, LatestStart a LatestFinish. Po výpočtu těchto vlastností se projdou všechny aktivity v projektu a zavolá se metoda CalculateOtherTimeAnalysisProperties, sloužící ke výpočtu jejich vlastností TotalTimeReserve, IsCritical, FreeTimeReserve a DependentTimeReserve (řádky 291 až 294). Následně se opět projdou všechny aktivity v projektu a zavolá se metoda CalculateTimeReserveDependetActivities (řádky 295 až 298). Na konec se projdou všechny cesty v projektu a zavolá se metoda CalculateTimeAnalysisProperties. Pokud je projekt stochastický, tak se na cestách volá i metoda CalculateStatisticValues (řádky 299 až 306). Na závěr se nastaví vlastnost TimeAnalysisCalculated na hodnotu true.

Ukázka kódu 28 - Metoda CalculateTimeAnalysis

```
286     private void CalculateTimeAnalysis()
287     {
288         SetEarliestStartAndFinishProperties();
289         SetLatestStartAndFinishProperties();
290 
291         foreach (var activity in Activities)
292         {
293             activity.CalculateOtherTimeAnalysisProperties();
294         }
295         foreach (var activity in Activities)
296         {
297             activity.CalculateTimeReserveDependentActivities();
298         }
299         foreach (var path in Paths)
300         {
301             path.CalculateTimeAnalysisProperties();
302             if (SolutionType == SolutionType.Stochastic)
303             {
304                 (path as StochasticPath).CalculateStatisticValues();
305             }
306         }
307 
308         TimeAnalysisCalculated = true;
309     }
```

(zdroj: Autor)

Metoda SetEarliestStartAndFinishProperties (Ukázka kódu 29) slouží k nastavení vlastností aktivit EarliestStart a EarliestFinish. Na začátku je nalezena aktivita představující začátek projektu a zavolána její metoda CalculateEarliestStartAndFinish (řádky 313 a 314). Poté je vytvořena fronta a je do ní tato aktivita přidána (řádky 316 a 317). Následně probíhá cyklus, dokud fronta není prázdná (319). V tomto cyklu je vyndána z fronty aktivita (řádek 321), projdou se všichni její následovníci (řádek 307). Všem následovníkům, kteří jsou proiterováni, je volána metodu CalculateEarliestStartAndFinish (řádek 324). Tato metoda vrací true, pokud se vlastnosti, které má tato metoda nastavit zvětšily. Pokud tomu tak je, tak je následovník přidán do fronty (řádek 326), čímž dojde k přepočtení vlastností EarliesStart a EarliestFinish všem jeho následovníkům. Je to z toho důvodu, že na stejnou aktivitu se lze dostat různými předchůdci. Jelikož vypočtené hodnoty této metody jsou založeny na vypočtených hodnotách jejích předchůdců, tak pro správné výpočet musí mít všichni předchůdci aktivity nastavené správné hodnoty.

Ukázka kódu 29 - Metoda SetEarliestStartAndFinishProperties

```
311  private void SetEarliestStartAndFinishProperties()
312  {
313      var startingActivity = Activities.First(u => u.IsStart);
314      startingActivity.CalculateEarliestStartAndFinish();
315
316      var queue = new Queue<Activity>();
317      queue.Enqueue(startingActivity);
318
319      while (queue.Count != 0)
320      {
321          var activity = queue.Dequeue();
322          foreach (var successor in activity.Successors)
323          {
324              if (successor.CalculateEarliestStartAndFinish())
325              {
326                  queue.Enqueue(successor);
327              }
328          }
329      }
330  }
```

(zdroj: Autor)

Metoda SetLatestStartAndFinishProperties (Ukázka kódu 30) slouží k nastavení vlastností aktivit LatestStart a LatestFinish. Funguje úplně stejně jako metoda SetEarliestStartAndFinishProperties, akorát se projekt prochází od konce. To znamená, že je do fronty na počátku metody přidána aktivita, která představuje konec projektu (řádek 338). A místo metody CalculateEarliestStartAndFinish je volána aktivitám metoda CalculateLatestStartAndFinishProperties. Změna je také v tom, že v těle cyklu se proiterovávají předchůdci aktivit (řádek 343). Přidáním předchůdce do fronty (řádek 347) dojde k přepočítání vlastností LatestStart a LatestFinish všem předchůdcům daného předchůdce. Je to z toho důvodu, že na stejnou aktivitu se lze dostat z více následovníků, a aby metoda ClaculateLatestStartAndFinish fungovala, musí mít všichni následovníci správně vypočítané vlastnosti LatestStart a LatestFinish.

Ukázka kódu 30 - Metoda SetLatestStartAndFinishProperties

```
332     private void SetLatestStartAndFinishProperties()
333     {
334         var endingActivity = Activities.First(u => u.IsEnd);
335         endingActivity.CalculateLatestStartAndFinish();
336
337         var queue = new Queue<Activity>();
338         queue.Enqueue(endingActivity);
339
340         while (queue.Count != 0)
341         {
342             var activity = queue.Dequeue();
343             foreach (var precessor in activity.Precessors)
344             {
345                 if (precessor.CalculateLatestStartAndFinish())
346                 {
347                     queue.Enqueue(precessor);
348                 }
349             }
350         }
351     }
```

(zdroj: Autor)

Metoda RemoveTransitivePaths (Ukázka kódu 31) vymaže z kolekce Paths všechny cesty obsahující tranzitivní vztah mezi aktivitami. Vymazány jsou konkrétně ty cesty, které přeskakují kvůli tranzitivitě nějaké aktivity (neobsahují je), ale mají jinak stejné vlastnosti jako cesta, která je obsahuje (viz. kapitola Kritická cesta). Podmínkou její správné funkčnosti je, že byla již vypočítána časová analýza. Na řádku 240 se vytvoří kolekce, do které se budou ukládat nalezené tranzitivní cesty. Následně se projdou všechny cesty (řádek 241), a v těle tohoto cyklu se znovu projdou všechny cesty (řádek 243). Pokud se řídící proměnné prvního cyklu a řídící proměnné druhého cyklu rovná vlastnost Length, a nerovná se jim počet aktivit, které obsahují (řádek 245), tak se vstupuje do podmínky. V této podmínce je další podmínka, která zjistí, zda aktivity řídící proměnné prvního cyklu jsou podmnožinou aktivit řídící proměnné druhého cyklu (řádek 248). Pokud tomu tak je, tak je řídící proměnná prvního cyklu přidána do tranzitivních cest (řádek 251). Následně jsou všechny tranzitivní cesty proiterovány (řádek 256) a vymazány z vlastnosti Paths (řádek 258).

Ukázka kódu 31 - Metoda RemoveTransitivePaths

```
238     private void RemoveTransitivePaths()
239     {
240         var transitivePaths = new List<Path>();
241         foreach (var firstPath in Paths)
242         {
243             foreach (var secondPath in Paths)
244             {
245                 if (firstPath.Length == secondPath.Length
246                             && firstPath.Activities.Count != secondPath.Activities.Count)
247                 {
248                     if (!firstPath.Activities.Except(secondPath.Activities).Any()
249                                 && !transitivePaths.Contains(firstPath))
250                     {
251                         transitivePaths.Add(firstPath);
252                     }
253                 }
254             }
255         }
256         foreach (var transitivePath in transitivePaths)
257         {
258             Paths.Remove(transitivePath);
259         }
260     }
```

(zdroj: Autor)

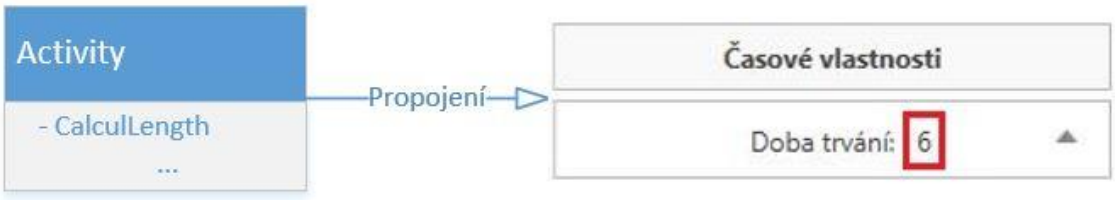
3.4 Propojení obrazovek aplikace s logickou částí

K propojení logické části aplikace s její vizuální stránkou byl použit návrhový vzor MVVM (Model-View-ViewModel). To znamená, že mezi třídami popsanými v předchozí kapitole a obrazovkami, které jsou popsány v následující kapitole, se nacházejí třídy

označované v angličtině jako ViewModels. Tyto třídy slouží k převedení některých dat z logické části aplikace do srozumitelné formy pro uživatele, která je mu následně prezentována.

Na obrázku 21 je zobrazen případ, kdy není potřeba formátovat data z logické třídy pro obrazovku aplikace zvláštním způsobem. Vše, co stačí udělat, je vytvořit propojení (anglicky binding), mezi danou vlastností z logické části třídy a kontrolkou, v které se má hodnota zobrazit. V kontrolce zobrazující dobu trvání aktivity (na obrázku 21 červený obdélník) je očekávána číselná hodnota. Ta je uložena ve vlastnosti CalculLength třídy Activity. Proto stačí jen tuto kontrolku s danou vlastností propojit a data není potřeba dále formátovat.

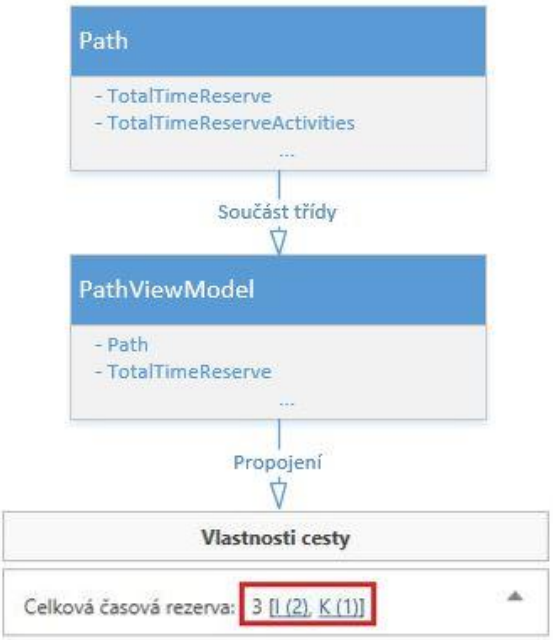
Obrázek 21 – Propojení třídy Activity a obrazovky bez využití ViewModel třídy



(zdroj: Autor)

Obrázek 22 ukazuje případ, kdy je potřeba formátovat data z logické třídy. Vlastnost TotalTimeReserve třídy Path uchovává údaj o celkové časové rezervě v datovém typu double. Vlastnost TotalTimeReserveActivites je kolekce referencí na aktivity, které se na vlastnosti TotalTimeReserve podílejí. Třída PathViewModel obsahuje vlastnost Path, v které je uložena reference na instanci třídy Path, ke které se daná instance třídy PathViewModel vztahuje. Vlastnost TotalTimeReserve třídy PathViewModel čerpá informace z vlastností TotalTimeReserve a TotalTimeReserveActivites třídy Path a ty dále formátuje, aby mohlo vzniknout propojení s kontrolkou zobrazující celkovou časovou rezervu cesty (na obrázku červený obdélník).

Obrázek 22 - Propojení třídy Path a obrazovky s využitím ViewModel třídy



(zdroj: Autor)

Popis tříd, které se nacházejí mezi logickými třídami aplikace a jednotlivými obrazovkami není součástí této bakalářské práce. Jsou však k dispozici k prohlédnutí na datovém nosiči, který byl dodán jako příloha této bakalářské práce.

3.5 Popis jednotlivých obrazovek aplikace

3.5.1 Okno aplikace a hlavní menu

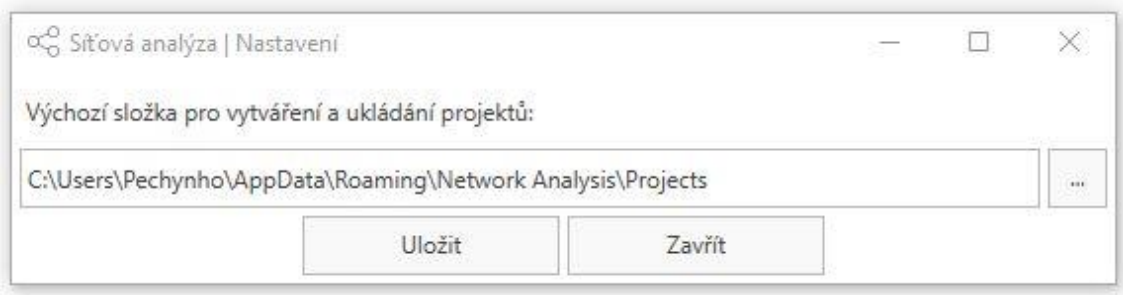
Okno aplikace (Příloha 1) se skládá z hlavního menu a prostoru pro informace o aplikaci a jednotlivých projektech. Ty jsou od sebe odděleny pomocí záložek. Každý nový projekt se otevře v nové záložce. Záložka s názvem Úvodní informace obsahuje informace o aplikaci a panel s naposledy otevřenými projekty, který je popsaný v následující kapitole. Záložka projektu nese označení podle názvu projektu, které zadá uživatel. Hlavní menu obsahuje tři položky. Těmi jsou Projekt, Nastavení a Jak na načtení z CSV. Po kliknutí na položku Projekt se otevře rozbalovací seznam obsahující položky:

- Nový projekt – Po kliknutí na tuto položku se uživateli otevře dialog, v kterém vybere název souboru, do kterého chce informace projektu ukládat, a následně ho program přesměruje do zadávání informací o projektu, které je popsáno dále v této práci.

- Otevřít existující… - Kliknutí na tuto položku otevře uživateli průzkumníka souborů, kde bude moci vybrat soubor typu NETAN, v kterém jsou uloženy informace o projektu.
- Načíst z CSV souboru – Tato položka umožňuje uživateli načíst projekt z CSV souboru, který musí mít daný formát.
- Uložit – Po kliknutí se uloží informace o projektu.
- Uložit jako… - Tato položka umožňuje uživateli také uložit informace o projektu, ale zeptá se ho, do jakého souboru chce data uložit.
- Otevřít složku – Otevře složku, v které se nachází soubor s projektem, s kterým uživatel právě pracuje.
- Smazat projekt – Po kliknutí smaže soubor, v kterém jsou uloženy informace o projektu, s kterým uživatel právě pracuje.
- Zavřít – Ukončí celý program. Pokud jsou otevřené nějaké projekty, tak se aplikace nejdříve uživatele zeptá, jestli chce uložit provedené změny.

Kliknutí na položku Nastavení se otevře dialog (Obrázek 23) v kterém uživatel může navolit složku, do které se mají primárně vytvářet nové projekty.

Obrázek 23 - Dialog s nastavením aplikace



(zdroj: Autor)

Po kliknutí na položku Jak na načtení z CSV se uživateli otevře okno s nápovědou, která říká, jaký formát musí mít data v CSV souboru, aby z něj bylo možné načíst projekt. Nacházejí se zde i ukázkové příklady.

3.5.2 Naposledy otevřené projekty

Úvodní obrazovka aplikace obsahuje panel, který zobrazuje naposledy otevřené projekty (Příloha 2). Uživatel tak má přístup k projektům, s kterými naposledy pracoval. Okamžitě vidí, o jaký druh projektu šlo, jeho název a kdy ho naposledy otevřel.

3.5.3 Zadávání informací o projektu

Tato obrazovka (Příloha 3) je jednou ze základních částí pohledů projektu. Zde se zadávají všechny informace o projektu. Kontrolkami nacházejícími se v horní části této obrazovky se upravuje druh projektu, může se měnit celkový počet aktivit v projektu, nebo je možno vyhledat aktivitu v tabulce níže. Pokud je potřeba přidat určitý počet aktivit, tak se zadá požadovaný počet a klikne se na tlačítko Přidat. Stejným způsobem se postupu při odebírání aktivit z projektu, jen se klikne na tlačítko Odebrat. Odebrat aktivity je možno také tak, že se zaškrtnou v tabulce uprostřed obrazovky, a následně se klikne na tlačítko Vymazat zaškrtlé. Vedle tabulky obsahující všechny aktivity se nachází panel, do kterého uživatel vyplňuje informace o aktivitě. Aplikací je upozorňován, pokud zadá nějaký nevalidní vstup (např. písmeno místo čísla). K jednoduchému přehledu o validaci zadaných informací o aktivitě slouží sloupeček Validace. Zde se objeví křížek, pokud některé informace zadané o aktivitě nejsou validní. Po zastavení myší nad křížkem se uživateli zobrazí přehled s nedostatky, které aktivita má. Tlačítko vypracovat analýzu je uživateli k dispozici ke stisknutí, až když zadá validní informace všem aktivitám v projektu. Panel pro zadávání informací o aktivitě mění svůj vzhled podle druhu projektu, protože pokaždé je potřeba získat trošku jiné informace od uživatele.

3.5.4 Síťový diagram

Časová analýza projektu nabízí uživateli tři pohledy. Jedním z nich je pohled ukazující síťový diagram projektu (Příloha 4). V tomto pohledu uživatel vidí provázání jednotlivých aktivit mezi sebou. Tento diagram je plně interaktivní. To znamená, že po kliknutí na aktivitu v diagramu se uživateli v dolním panelu ukáží informace o vybrané aktivitě. Síťový diagram i panel s informacemi může uživatel libovolně schovat a opět zobrazit pomocí tlačítka s šipkou. Diagram lze pomocí posuvníku pod ním libovolně zvětšit nebo zmenšit. Spodní panel s informacemi má dvě záložky. První záložka s názvem Aktivity zobrazuje informace o jednotlivých aktivitách v projektu. Jsou zde vidět tři základní bloky a těmi jsou Základní vlastnosti, Vztahy s ostatními aktivitami a Časové vlastnosti. Všechny popisky, které mají modrou barvu, a jsou modře podtrženy, slouží jako odkazy. Pokud se jedná o odkaz na aktivitu, tak se uživateli zobrazí informace o dané aktivitě. Odkaz může být i na cestu v projektu. V takovém případě uživatele navede na informace dané cestě.

3.5.5 Ganttův diagram

Dalším pohledem časové analýzy (Příloha 5), kterou aplikace poskytuje je Ganttův diagram (Příloha 6). Ovládání diagramu i panel s informacemi pod diagramem jsou totožné jako u obrazovky zobrazující síťový diagram. Aktivity, které mají časovou rezervu, jsou znázorněný pomocí posuvníku. Uživatel může aktivitu po posuvníku táhnout, a tím čerpat její časovou rezervu. Pokud se čerpá podmíněná časová rezerva aktivity, tak se posouvají i aktivity, podílející se na této podmíněné časové rezervě. Oproti příloze zobrazující síťový diagram (Příloha 4) je v příloze 5 v panelu s informacemi vybraná záložka Cesty. V tomto panelu uživatel může měnit název cesty, prohlédnout si informace o všech cestách v projektu, nebo si cestu zvýraznit. Je zde i blok umožňující zobrazení informací o části cesty. Stačí vybrat jen hraniční aktivity.

3.5.6 Tabulkový přehled

Posledním pohledem společným pro oba druhy projektů je obrazovka s tabulkovým přehledem (Příloha 7). Zde má uživatel všechny informace o aktivitách v projektu zapsané do tabulky. Pod tabulkou s aktivitami se nachází panel s informacemi o cestách v projektu. Jeho vzhled i funkčnost je stejný jako u panelu Cesty popsaného v tématu Ganttův diagram. Celou tabulku s informacemi si uživatel může vyexportovat do CSV souboru (Příloha 8), který může otevřít například v programu Microsoft Excel a s informacemi provádět své další analýzy a výpočty. Spodní panel lze skrýt pomocí tlačítka s šipkou. Tlačítko Skrýt sloupce cest schová sloupce zobrazující, jestli se aktivita nachází v dané cestě, nebo ne. Pokud jsou tyto sloupce schované, tak nejsou ani součástí exportu do CSV souboru.

3.5.7 Pravděpodobnostní kalkulátor

Speciálním pohledem pro stochastické projekty je pravděpodobnostní kalkulátor (Příloha 9). Zde uživatel může zadávat 4 druhy výpočtů vztahující se k otázkám popsaných v kapitole Vypořádání se s nejistým trváním aktivit. Při výpočtu odpovědi pro celý projekt aplikace odstraňuje první zjednodušující předpoklad popsaný ve výše zmíněné kapitole a pravděpodobnostní výpočet provádí druhým postupem, který je také popsaný ve výše zmíněné kapitole. Uživatel si může prohlédnout jak matematické zadání a odpověď, tak i slovní zadání a odpověď. Dále se na této obrazovce nacházejí dva panely, v kte-

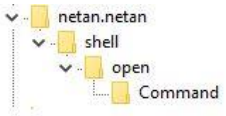
rých si uživatel může vybrat libovolnou cestu a aplikace pro něj vypočte výsledky konkrétně pro tuto vybranou cestu. V pravé části obrazovky jsou zobrazeny pravděpodobnostní grafy pro vybrané cesty ze zmíněných panelů.

3.6 Vlastní souborový formát

Aplikace využívá k uchování dat vlastní souborový formát. Jeho princip byl již nastíněn v kapitole Třída Project. Informace o projektu jsou pomocí binární serializace uloženy do složky, která je komprimována pomocí kompresního formátu ZIP a koncovka takto zabalené složky je nastavena na NETAN. Aby při otevření souboru s příponou NETAN operační systém Windows volal naší aplikaci, je potřeba zapsat několik informací do registru Windows. Aplikaci bude cesta k souboru předána pomocí počátečních argumentů. S touto hodnotou může aplikace dále pracovat a data načíst. Potřebné hodnoty se budou zapisovat do registru HKEY_CLASSES_ROOT, v kterém jsou uchovány informace o příponách souborů. Z tohoto registru mohou aplikace defaultně jen číst. K zápisu je potřeba zvýšených práv. Jelikož však níže popsaná metoda je volána v instalátoru aplikace, tak máme jistotu, že práva k zápisu aplikace obdrží.

Pro volání aplikace při otevírání souborů s příponou NETAN je nejdříve nutné zapsat do registru HKEY_CLASSES_ROOT nový podklíč ve tvaru netan.netan. Před tečkou je zapsána přípona aplikace, a za tečkou je unikátní klíč aplikace. Takto vytvořenému podklíči se vytvoří další podklíč s hodnotou "shell". Tomuto podklíči se vytvoří podklíč "open" a jemu se vytvoří podklíč "Command". Podklíči "Command" se nastaví hodnota ve tvaru: cestka k aplikace + "%1". Na mém počítači tímto způsobem vytvořená hodnota vypadá takto: C:\Program Files (x86)\Síťová analýza\NetworkAnalysis.exe "%1". Na obrázku 24 je vyobrazena výše popsaná hierarchie klíčů zobrazená pomocí editoru registru Regedit.

Obrázek 24 – Výsledná hierarchie klíčů v registru



(zdroj: Autor)

Metoda Register (Ukázka kódu 32) zajišťuje průběh výše popsaného postupu. Metodě je předána cesta k EXE souboru aplikace. Nejdříve se na řádku 23 otevře registr HKEY_CLASSES_ROOT. Následně se v něm vytvoří všechny potřebné podklíče (řádky

25 až 28) a poslednímu podklíči se nastaví hodnota ve správném tvaru (řádek 30) (“IT-network.cz”, 2017).

Ukázka kódu 32 - Metoda Register

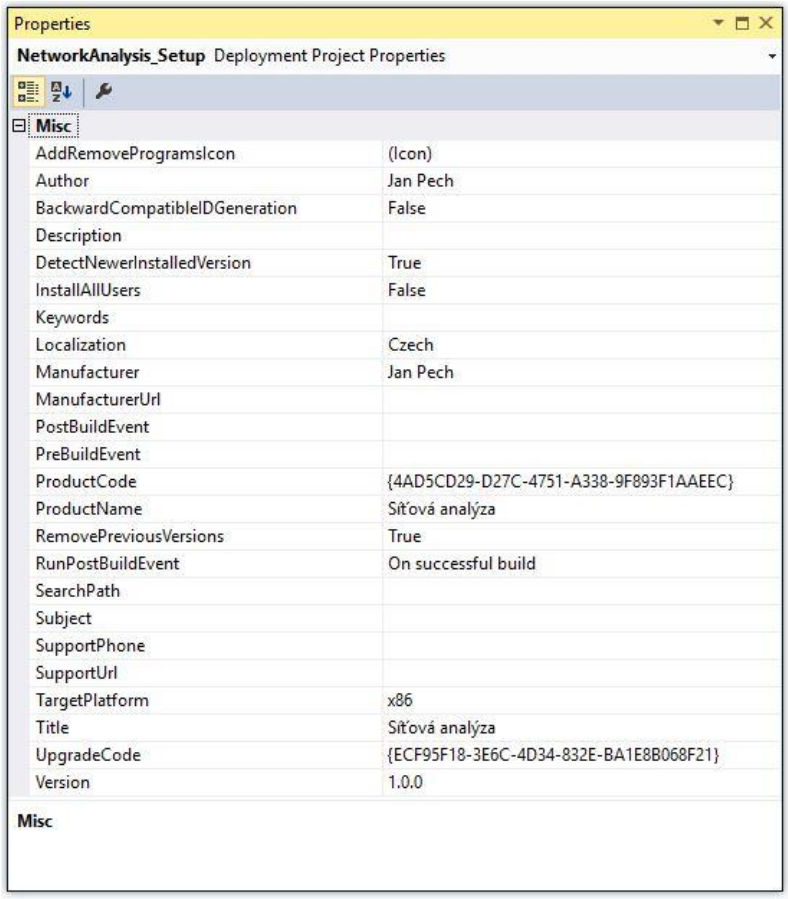
```
21  public void Register(string pathToApp)
22  {
23      var classes = Registry.ClassesRoot;
24
25      var extension = classes.CreateSubKey(this.extension + ".netan");
26      var shell = extension.CreateSubKey("shell");
27      var open = shell.CreateSubKey("open");
28      var command = open.CreateSubKey("Command");
29
30      command.SetValue("", pathToApp + " \"%1\"");
31  }
```

(zdroj: Autor)

3.7 Instalátor aplikace

K úspěšnému vydání aplikace je potřeba vyřešit způsob, jak bude šířena mezi uživatele. Visual Studio nabízí několik způsobů, kterými lze instalace aplikací vyřešit. Jedním z nich je tzv. Setup Project. Tento způsob vyřeší instalaci, odinstalaci a aktualizaci aplikace. Stačí jen do projektu s aplikací přidat nový projekt a ze seznamu možností vybrat Setup Project. Instalace je prováděna pomocí souboru s příponou MSI, do kterého Setup Project zabalí všechny potřebné soubory a knihovny, které aplikace potřebuje. Sám tyto soubory a knihovny rozpozná. MSI je instalační balíčkovací souborový formát, který využívají výhradně operační systémy Windows. Po přidání Setup Project do projektu je důležitá k jeho správné funkčnosti jediná věc, a to vyplnit všechny informace v okně Properties, jako je znázorněno na obrázku 25. Zajímavé je pole Product Code, jehož hodnotu Visual Studio automaticky vygeneruje. Tento kód slouží k rozlišení jednotlivých nainstalovaných programů od sebe. Pokud je žádoucí vydat novou verzi aplikace (např. 1.0.1), tak Visual Studio vygeneruje nový Product Code, aby se jednotlivé verze od sebe odlišily. Je však důležité, aby obě verze (z které a na kterou se aktualizuje) měly stejný Upgrade-Code. Nyní stačí jen sestavit SetupProject a vytvoří se MSI soubor, který lze použít k instalaci. Odinstalace je prováděna pomocí panelu Programy a funkce v systému Windows.

Obrázek 25 – Okno Properties patřící Setup Projectu¨



(zdroj: Autor)

Může nastat situace, že je potřeba udělat nějakou vlastní akci při instalaci aplikace. V případě této aplikace je to zapsání do registru operačního systému Windows pro vytvoření asociace koncovky NETAN. K tomu stačí do projektu přidat nový projekt, a ze seznamu možností vybrat tzv. InstallerClass. V této třídě lze následně přepsat metodu Install a provést vlastní akce (Ukázka kódu 33). Důležité je zavolat akci, která provede instalaci aplikace (zapsání potřebných hodnot do registru, nakopírování potřebných souborů do složek atd.). To se provede zavoláním rodičovské metody Install (řádek 43), které se předá proměnná stateSaver. Následně je možno provádět vlastní akce. V bloku na odchytávání výjimek se vytvoří nová instance třídy umožňující registraci koncovky (řádek 47). Následně se zjistí, zda koncovka NETAN již není registrována zavoláním metody IsRegistered (řádek 48). Pokud tomu tak není, volá se metodu Register (řádek 50), která je vysvětlena v kapitole Vlastní souborový formát. Této metodě je předána cesta k EXE souboru aplikace, k níž se dostaneme způsobem, který je vidět na zmíněném obrázku.

Ukázka kódu 33 - Metoda Install

```
41  public override void Install(IDictionary stateSaver)
42  {
43      base.Install(stateSaver);
44
45      try
46      {
47          var extensionRegistration = new ExtensionRegistration();
48          if (!extensionRegistration.IsRegistred())
49          {
50                extensionRegistration.Register(this.Context.Parameters["assemblypath"]);
51          }
52      }
53      catch (Exception ex)
54      {
55          MessageBox.Show(ex.Message);
56      }
57  }
```

(zdroj: Autor)

3.8 Dekompilace a obfuskace zdrojového kódu

Jelikož je C# interpretovaný programovací jazyk, tak jeho dekompilace je velmi jednoduchá. Zdrojový kód napsaný pomocí jazyka C# není převáděn přímo do strojového kódu, který je pro lidské oko nečitelný a jeho dekompilace zpátky do zdrojového kódu je velmi obtížná. Místo toho je převáděn do CIL (Common Intermediate Language), což je programovací jazyk. Ke spuštění tohoto programovacího jazyka je potřeba mít nainstalovaný tzv. interpreter, který kód v tomto jazyce provádí (interpretuje). Tento přístup má své výhody (dynamické typování, možnost reflexe, …) i nevýhody (pomalejší chod programu, nutnost mít nainstalovaný potřebný interpreter, …). Jednou z největších nevýhod je, že CIL uchovává o původním zdrojovém kódu velmi mnoho informací a dekompilace zpátky do zdrojového kódu je velmi jednoduchá. Na internetu jsou lehce dostupné programy, které toto umožňují, a program dekompilují téměř okamžitě (např. program dotPeek). K ochraně zdrojového kódu mohou vést různé důvody. Ať už se jedná o ochranu duševního vlastnictví (algoritmy společností pro různé procedury), ochranu přihlašovacích údajů k databázím uložené přímo v kódu, nebo skrytí metod užitých k šifrování informací.

Jednou z možných ochran před dekompilací kódu je tzv. obfuskace zdrojového kódu. To znamená, že se zdrojový kód udělá, co nejhůře čitelný pro lidské oko. Při vývoji však je požadováno, aby zdrojový kód byl co nejlépe čitelný kvůli jeho další snadné udržitelnosti. Proto je nejlepší, aby k obfuskaci došlo během převodu zdrojového kódu do CIL. K tomu lze využít velké množství programů. Já jsem si vybral aplikaci Crypto Obfuscator For .NET. Tato aplikace umožňuje velké množství způsobů, jak zdrojový kód udělat

co nejvíce nečitelný. Umožňuje například přejmenovat názvy všech proměnných, vlastností, metod, tříd aj. na netisknutelné neviditelné znaky, které následně vypadají pro lidské oko stejně. Dále umožňuje skrytí volání metod, maskování běžných programovacích vzorů (cykly, rozbalování, přetěžování, podmínky. …) a mnoho jiných funkcí. Na obrázku 26 je vidět zmíněné maskování běžných programovacích vzorů, kdy kód před obfuskací (na obrázku vlevo) a po obfuskaci (na obrázku vpravo) je diametrálně odlišný, a po obfuskaci je pro člověka daleko hůře čitelný.

Obrázek 26 – Ukázka obfuskace

```
private static void SampleMethod(object[] o)
{
   if (o != null)
   {
      string[] result = new string[o.Length];
      for (int i = 0; i < o.Length; i++)
      {
         if ((o[i] is string) || o[i].GetType().Equals(typeof(StringBuilder)))
         {
            result[i] = i.ToString() + ": " + o[i];
         }
         else
         {
            result[i] = i.ToString() + ": null";
         }
      }
   }
}
```

```
private static void SampleMethod(object[] o)
{
   if (o != KEB.D)
   {
      string[] strArray = PAB.D((int) BMB.D(o));
      for (int i = 0; i < ((int) BMB.D(o)); i++)
      {
         if ((PZ.D(o[i]) != null) || o[i].GetType().Equals(Type.GetTypeFromHandle(KNB.D())))
         {
            strArray[i] = TGB.D(i.ToString(), ": ", o[i]);
         }
         else
         {
            strArray[i] = IZ.D(i.ToString(), ": null");
         }
      }
   }
}
```

(zdroj: http://www.ssware.com/cryptoobfuscator/features.htm)

4 Závěr

Metody pro řízení projektů PERT i CPM jsou postavené na jednoduchých myšlenkách a algoritmech, ale manažerům poskytují mocný nástroj pro řízení projektů. Při plánování a řízení projektů však velkou roli hraje čas a aplikace těchto metod na projekt bez využití výpočetní techniky je zdlouhavá. Navíc umění vedení projektů nezávisí na tom, jestli manažer zná metody PERT a CPM do nejpodrobnějších detailů, a umí je ručně spočítat. Hlavní je, aby uměl informace, které tyto metody poskytují, správně využít ke zdárnému splnění projektu. Proto je žádoucí, aby k získání těchto informací využil nějaký dostupný výpočetní software.

Praktický výstup této bakalářské práce je aplikace, která může posloužit manažerům při řízení projektu. Poskytuje kompletní informace, které lze získat z časové analýzy projektu. Její hlavní předností je jednoduché ovládání, tudíž ji mohou využít i uživatelé, kteří nejsou při práci s počítačem příliš zdatní. Stačí jim základní povědomí o tom, jaké informace metody CPM a PERT poskytují a základní informace o aktivitách v projektu. Uživatelům poskytuje ukládání informací pomocí vlastního souborového formátu, jednoduchou instalaci pomocí pár kliknutí, načítání projektů z CSV souborů a export výsledků časové analýzy také do CSV souboru. Aplikace projekty vizualizuje pomocí síťového diagramu, nebo Ganttovo diagramu. Uživatel si tak může vybrat, který způsob vizualizace preferuje. Při vývoji aplikace byl kladen velký důraz na validaci všech dat, která uživatel zadá. Pokud zadá nevalidní data, tak je na ně aplikací upozorněn a je mu řečeno, jaké vstupy jsou v tom konkrétním políčku očekávány.

V další verzích této aplikace bych chtěl přidat možnost vypracování časově-zdrojové a časově-nákladové analýzu projektu. Poté by tato aplikace představovala opravdu komplexní a mocný nástroj pro plánování projektů a mohla by v některých aspektech sekundovat daleko dražším komerčním nástrojům

5 Summary and keyword

The main target of this bachelor thesis is to develop an application suitable for project planning based on the network analysis.

The theoretical part of the thesis presents appropriate terminology and basic characteristics of the network analysis which uses information from the graph theory implemented in the planning of projects. The network analysis deals primarily with project time management which is described by the CPM method in deterministic projects and the PERT method in stochastic projects. Furthermore, it also tackles project resource management and project cost management.

The practical part describes the development of the application using the programming language C# and .NET technologies. The main purpose of the application is to help project managers, after entering necessary input information about a particular project to the application, to plan their projects in a sophisticated way and provides them with processed results in a synoptic graphical form.

Key words: network analysis, CPM, PERT, application development, C# .NET, project

6 Seznam použitých zdrojů

Hillier, F. S., & Lieberman, Gerald J. (2014). Introduction to Operations Research. Boston: McGraw-Hill.

Jablonský, J. (2007). Operační výzkum: kvantitativní modely pro ekonomické rozhodování. 3. vyd. Praha: Professional.

Friebelová, J. Operační analýza. České Budějovice: Jihočeská univerzita v Českých Budějovicích, Ekonomická fakulta.

Microsoft. (2015). Microsoft Solver Foundation 3.1. Microsoft Developer Network: MSDN Library [online]. 2016.03.15 [cit. 2016-03-29]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff524509

Nathan, A. (2013). WPF 4.5 Unleashed. Indianapolis, IN, USA: Sams.

Troelsen, A., & Japikse, P. (2015). C# 6.0 and the .NET 4.6 Framework. 7. vydání. New York, USA: Apress

Algoritmy.net. (2016). Algoritmy.net [Online]. Retrieved April 03, 2017, from https://www.algoritmy.net/article/1399/Prohledavani-do-sirky

ITnetwork.cz. (2017). ITnetwork.cz [Online]. Retrieved April 03, 2017, from http://www.itnetwork.cz/csharp/soubory/csharp-prace-s-vlastnimi-soubory-registry-uac-ikona

Stack Overflow. (2017). Stack Overflow [Online]. Retrieved April 03, 2017, from https://stackoverflow.com/

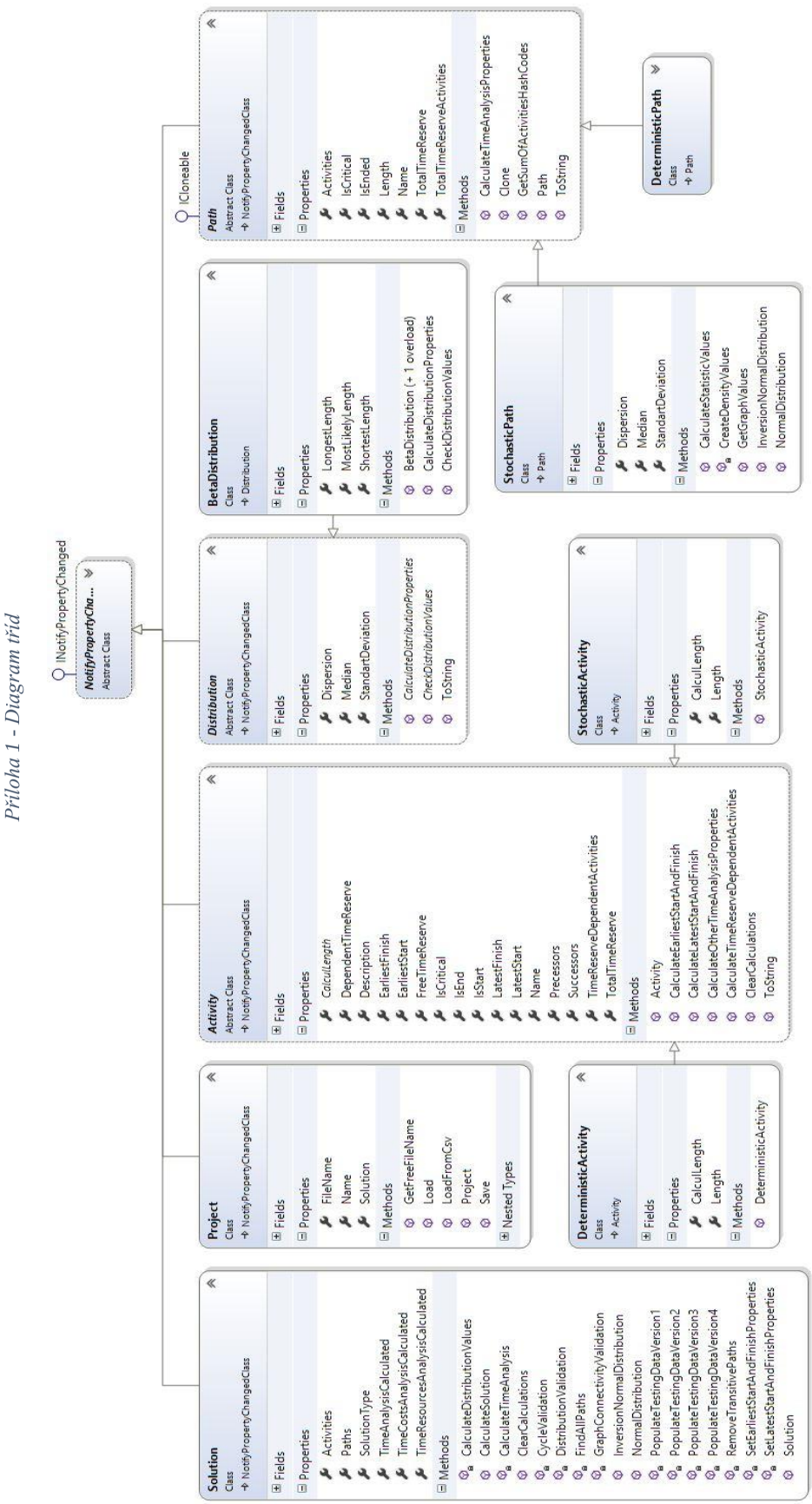
7 Seznam obrázků

8 Seznam tabulek

9 Seznam ukázek zdrojových kódů

10 Seznam příloh

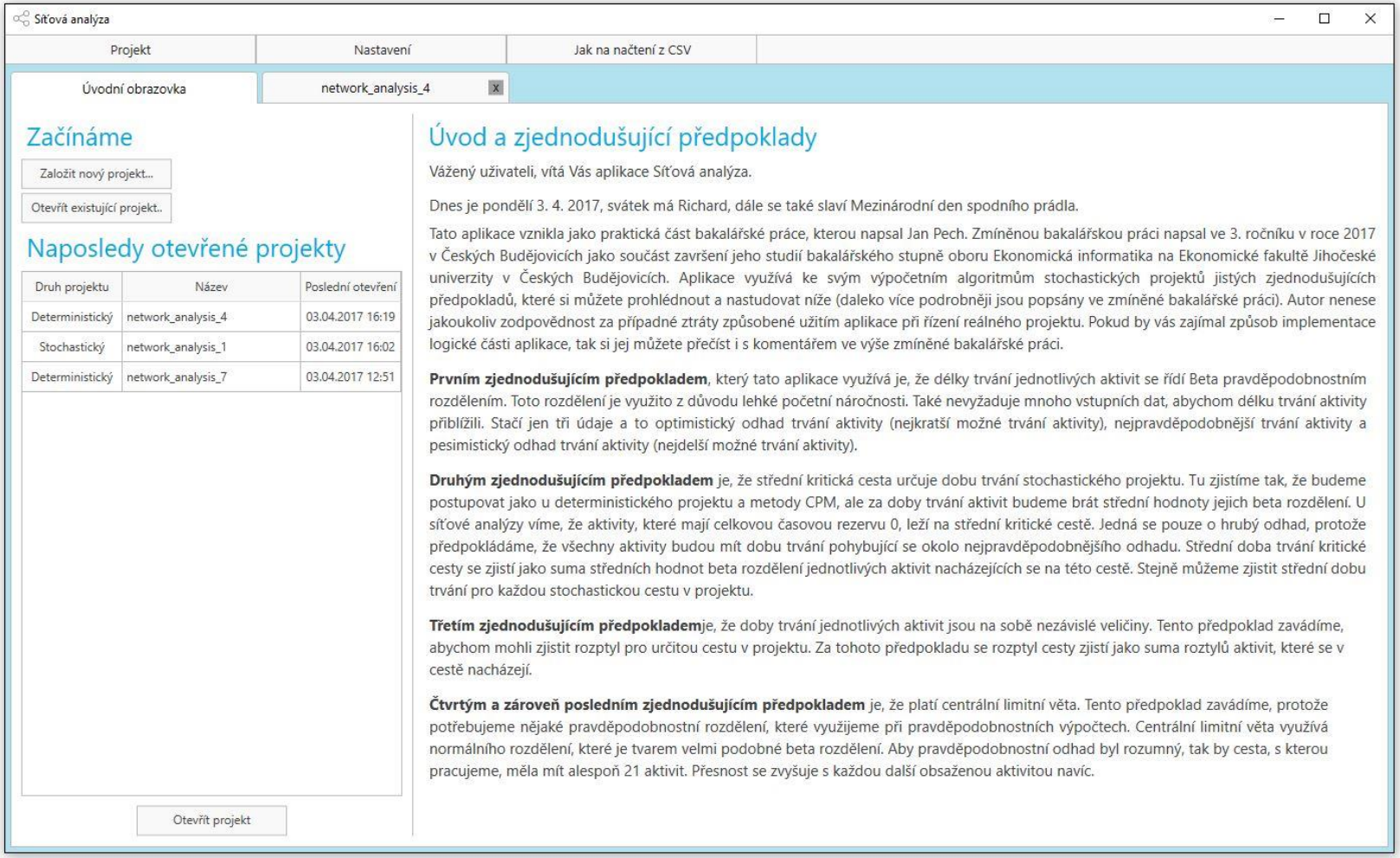
11 Přílohy



Příloha 1 - Diagram tříd

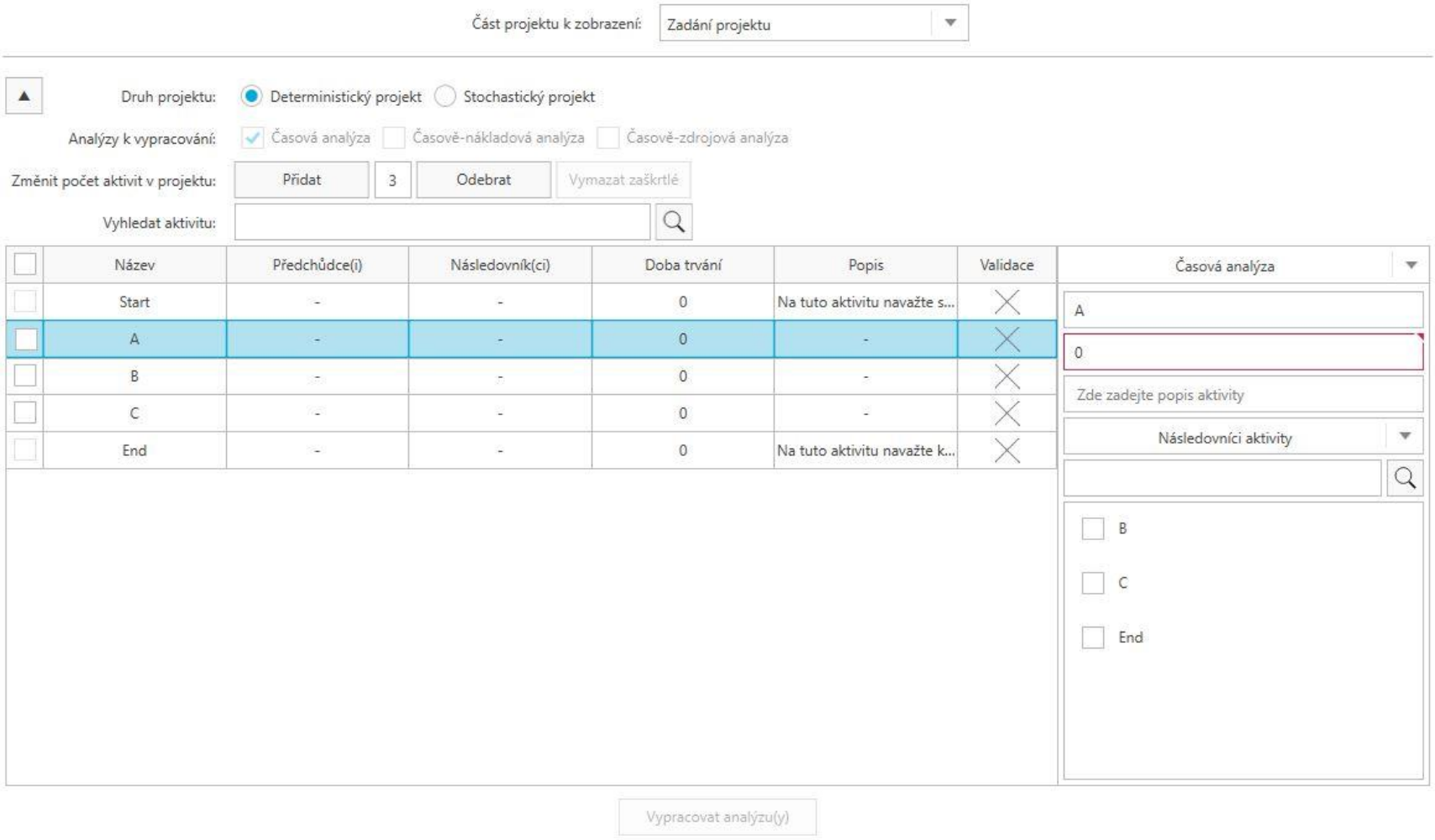
(zdroj: Autor)

Příloha 2 - Okno aplikace s úvodní obrazovkou



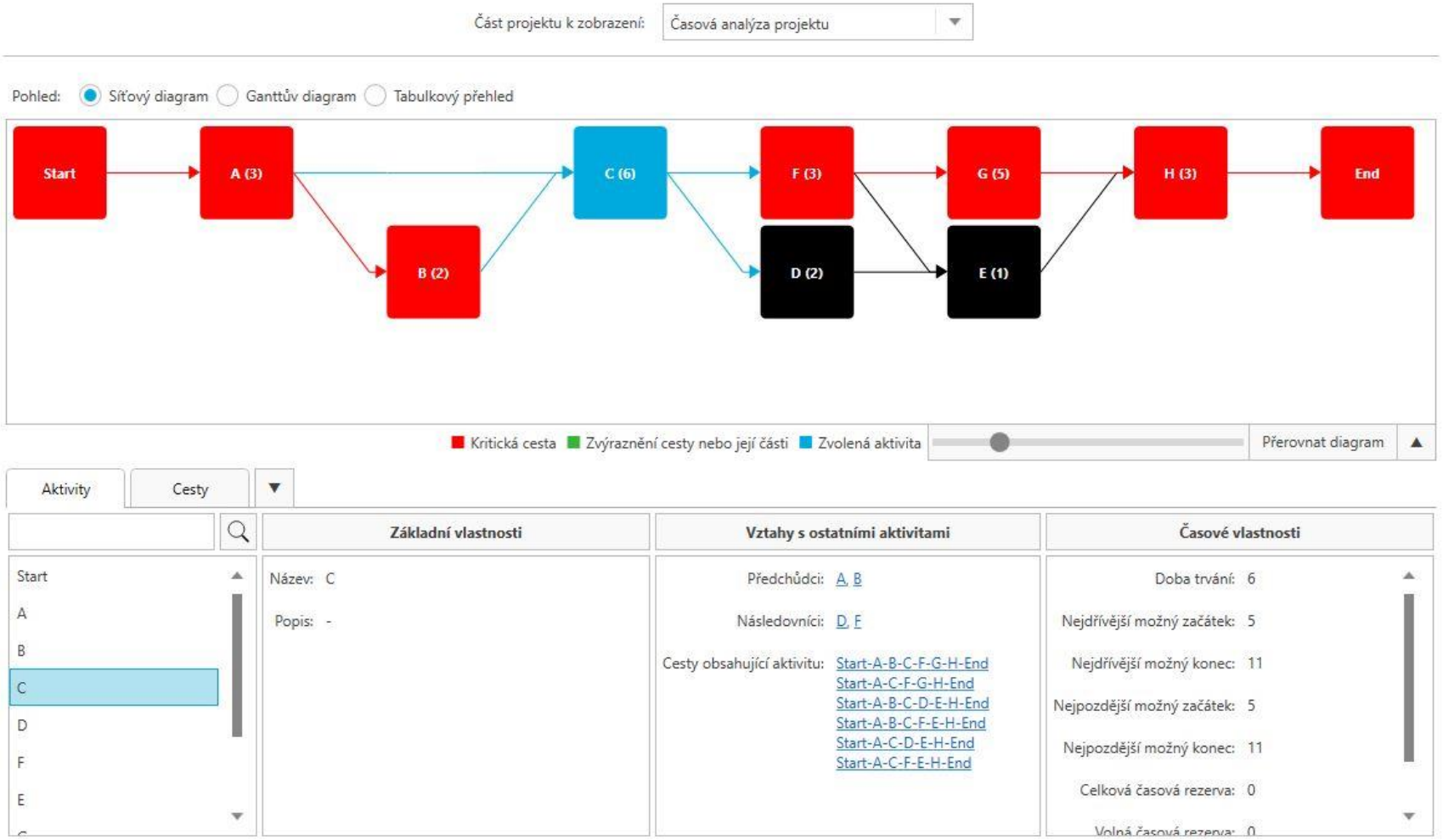
(zdroj: Autor)

Příloha 3 - Obrazovka se zadáváním informací o projektu



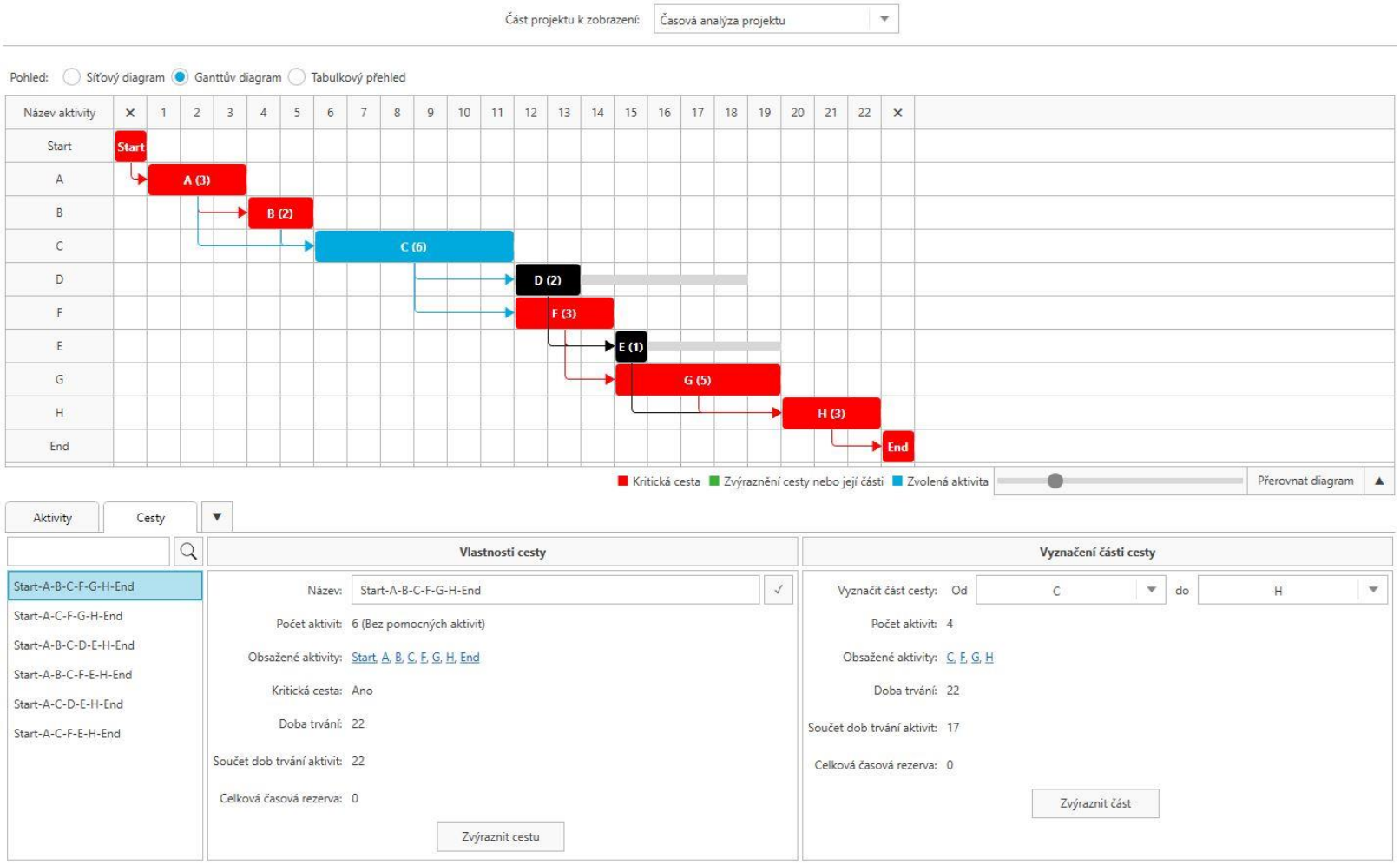
(zdroj: Autor)

Příloha 4 - Obrazovka se síťovým diagramem projektu



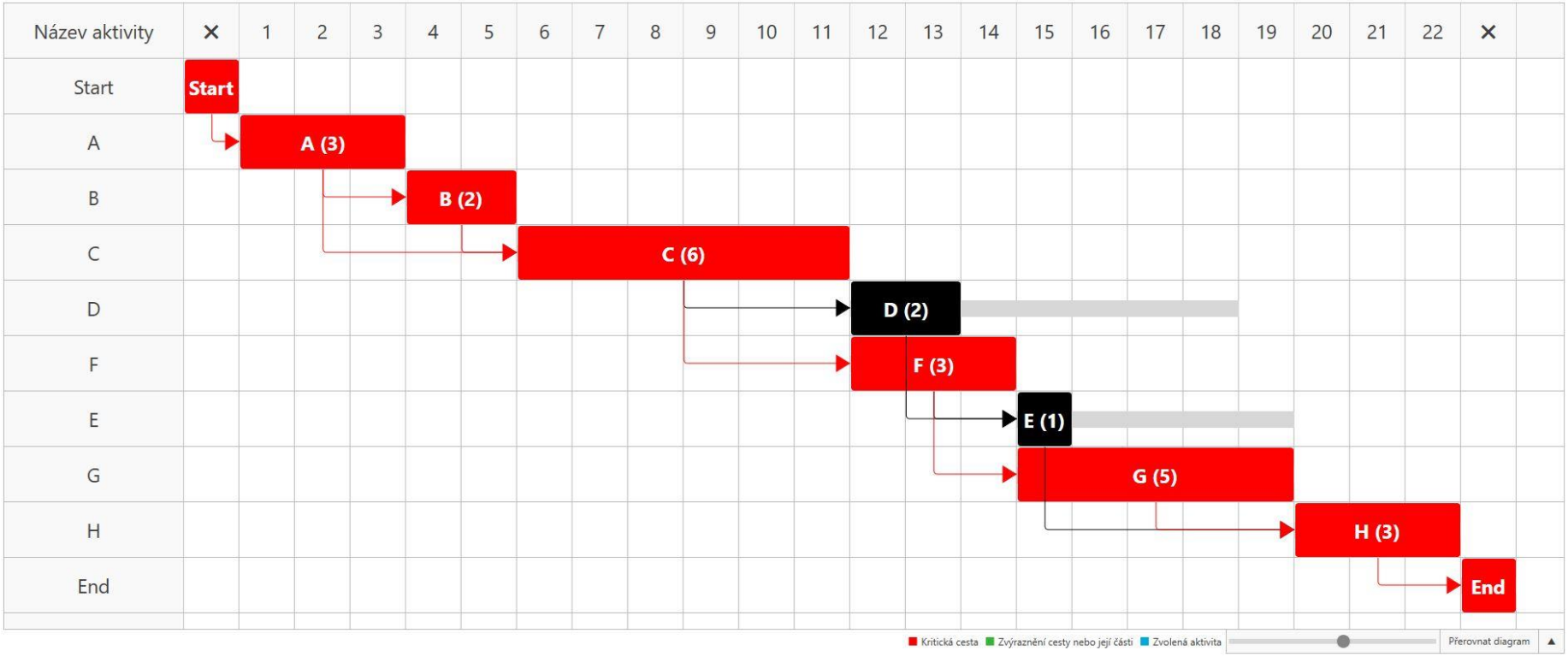
(zdroj: Autor)

Příloha 5 - Obrazovka zobrazující Ganttův diagram



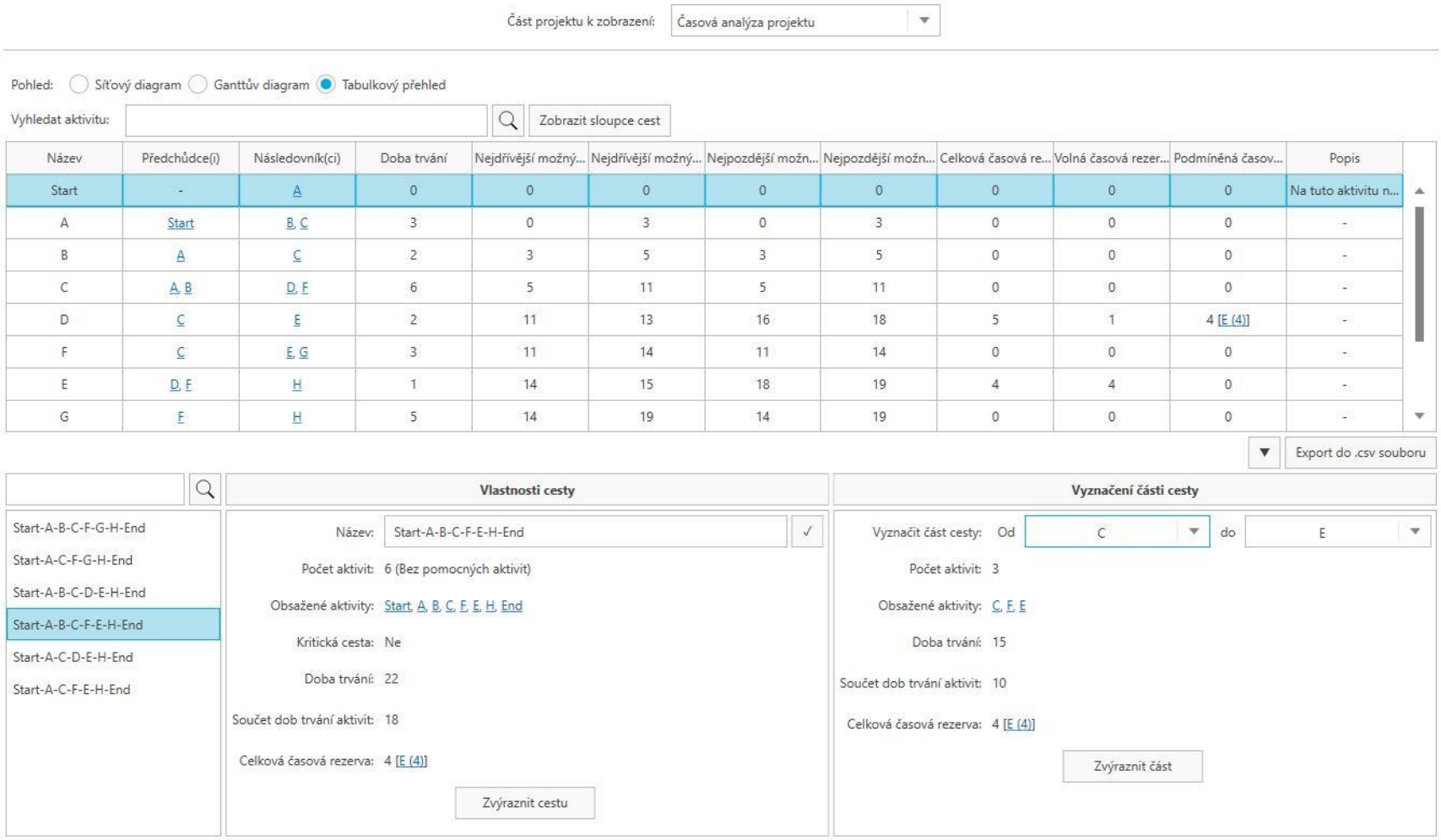
(zdroj: Autor)

Příloha 6 - Detail Ganttovo diagramu



(zdroj: Autor)

Příloha 7 - Obrazovka s tabulkovým přehledem



Název	Předchůdce(i)	Následovník(ci)	Doba trvání	Nejdřívější možný...	Nejdřívější možný...	Nejpozdější možn...	Nejpozdější možn...	Celková časová re...	Volná časová rezer...	Podmíněná časov...	Popis
Start	-	A	0	0	0	0	0	0	0	0	Na tuto aktivitu n...
A	Start	B, C	3	0	3	0	3	0	0	0	-
B	A	C	2	3	5	3	5	0	0	0	-
C	A, B	D, F	6	5	11	5	11	0	0	0	-
D	C	E	2	11	13	16	18	5	1	4 [E (4)]	-
F	C	E, G	3	11	14	11	14	0	0	0	-
E	D, F	H	1	14	15	18	19	4	4	0	-
G	F	H	5	14	19	14	19	0	0	0	-

(zdroj: Autor)

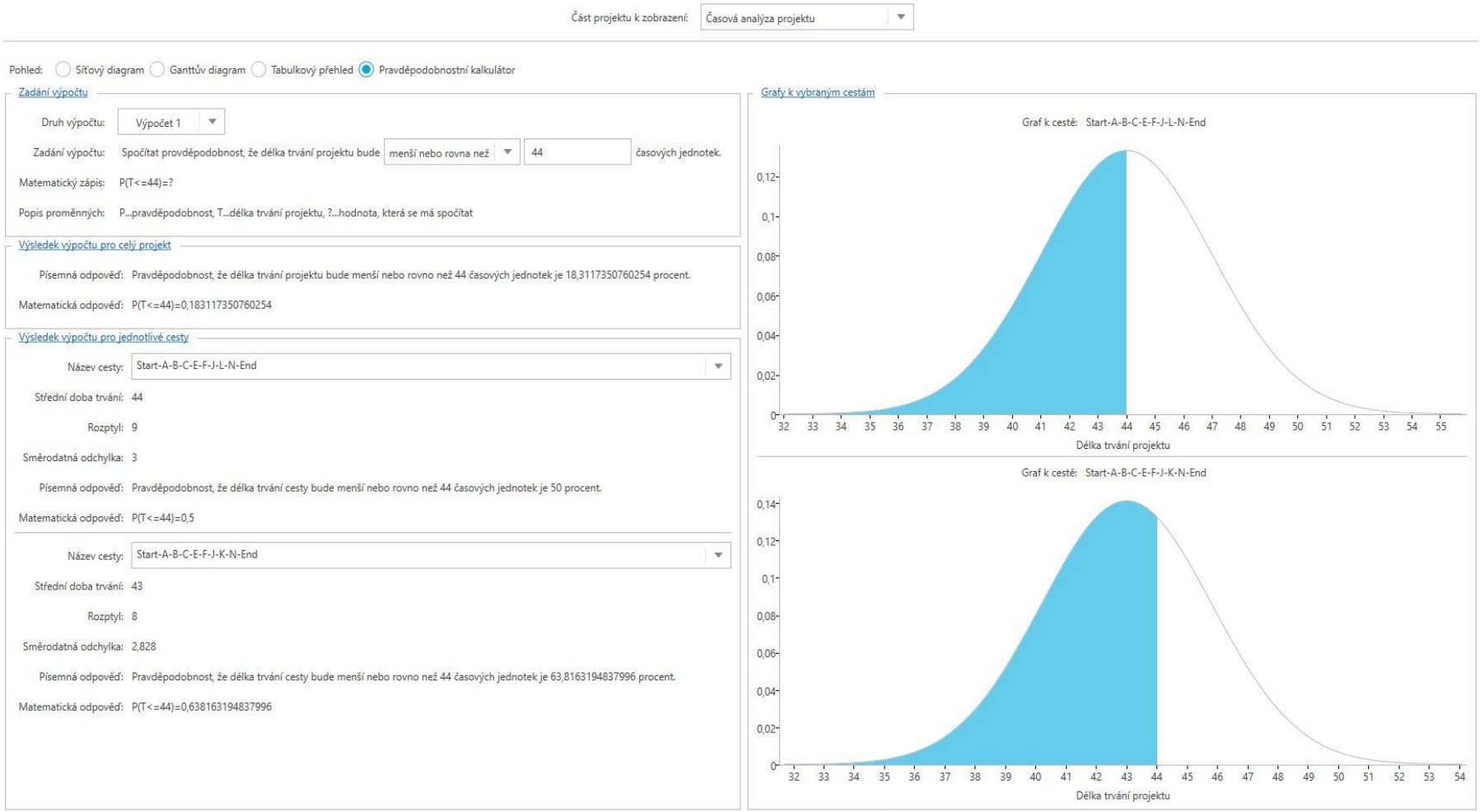
Příloha 8 - Ukázka exportovaného tabulkového přehledu

Název	Předchůd	Následovn	Optimistic	Nejpravdě	Pesimistic	Střední dc	Rozptyl	Směrodat	Nejdřívější	Nejdřívější	Nejpozdě	Nejpozdě	Celková č	Volná čas	Podmíněn	Popis
Start		A	0	0	0	0	0	0	0	0	0	0	0	0	0	Na tuto aktivitu navažte startovní aktivity(u) projektu.
A	Start	B	1	2	3	2	0,111111	0,333333	0	2	0	2	0	0	0	
B	A	C	2	3,5	8	4	1	1	2	6	2	6	0	0	0	
C	B	D, E, I	6	9	18	10	4	2	6	16	6	16	0	0	0	
E	C	H, F	1	4,5	5	4	0,444444	0,666667	16	20	16	20	0	0	0	
D	C	G	4	5,5	10	6	1	1	16	22	20	26	4	0	4	
I	C	J	3	7,5	9	7	1	1	16	23	18	25	2	2	0	
F	E	J	4	4	10	5	1	1	20	25	20	25	0	0	0	
G	D	H	5	6,5	11	7	1	1	22	29	26	33	4	0	4	
J	I, F	K, L	3	9	9	8	1	1	25	33	25	33	0	0	0	
H	E, G	M	5	8	17	9	4	2	29	38	33	42	4	0	4	
K	J	N	4	4	4	4	0	0	33	37	34	38	1	1	0	
L	J	N	1	5,5	7	5	1	1	33	38	33	38	0	0	0	
M	H	End	1	2	3	2	0,111111	0,333333	38	40	42	44	4	4	0	
N	K, L	End	5	5,5	9	6	0,444444	0,666667	38	44	38	44	0	0	0	
End	M, N		0	0	0	0	0	0	44	44	44	44	0	0	0	Na tuto aktivitu navažte konečné(ou) aktivity(u) projektu.

Název	Počet akti	Obsažené	Kritická ce	Střední dc	Součet stř	Rozptyl	Střední hc	Celková časová rezerva
Start-A-B-	10	Start, A, B	True	44	44	9	3	0
Start-A-B-	10	Start, A, B	False	44	43	8	2,828427	1
Start-A-B-	9	Start, A, B	False	44	41	7,555556	2,748737	3
Start-A-B-	9	Start, A, B	False	44	42	8,555556	2,924988	2
Start-A-B-	9	Start, A, B	False	40	40	11,22222	3,349959	4
Start-A-B-	8	Start, A, B	False	40	31	9,666667	3,109126	4

(zdroj: Autor)

Příloha 9 – Obrazovka s pravděpodobnostním kalkulátorem



(zdroj: Autor)