TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Uživatelské rozhraní pro simulátor BOModel

Bakalářská práce

Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy

Autor práce: **Victor Trnka**
Vedoucí práce: doc. Ing. Otto Severýn, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Uživatelské rozhraní pro simulátor BOModel

Jméno a příjmení: **Victor Trnka**
Osobní číslo: M17000061
Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Seznamte se se základy simulace vícefázového podzemního proudění a práce se simulátorem BOModel a jeho pomocných programů.
2. Navrhněte grafické uživatelské rozhraní, které umožní komfortnější práci uživatele s tímto simulátorem. Rozhraní by mělo obsahovat tyto funkcionality:
   - Automatické zálohování různých variant výpočtu a jejich výsledků, s možností návratu k předchozím řešením.
   - Spouštění simulátoru, řešení případných chyb při simulaci.
   - Editace datového souboru.
   - Spouštění vizualizačních nástrojů.
   - Porovnání výsledků různých variant výpočtu.
3. Proveďte rešerši existujících volně dostupných programů a knihoven vhodných pro vytvoření uživatelského rozhraní navrženého v bodě 2.
4. Pomocí vybraných softwarových prostředků navržené rozhraní implementujte a otestujte.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah pracovní zprávy: 30–40 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] PEACEMAN, Donald W. *Fundamentals of numerical reservoir simulation*. Elsevier Scientific Pub. Co. : distributors for the U.S. and Canada, Elsevier North-Holland, 1977. ISBN 978-0-444-41578-3.
[2] GEOQUEST ltd. Eclipse 100 User Guide, Slb, 2003.
[3] Dokumentace softwarových nástrojů používaných pro ložiskovou simulaci na pracovišti Geo-Services společnosti innogy Gas Storage s.r.o.

Vedoucí práce: doc. Ing. Otto Severýn, Ph.D.
Ústav mechatroniky a technické informatiky

Datum zadání práce: 10. října 2019
Předpokládaný termín odevzdání: 18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2019

Prohlášení

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

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

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

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

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

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

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

26. května 2020 Victor Trnka

Abstrakt

Práce se věnuje navržení a zrealizování grafického uživatelského prostředí pro práci se simulátorem vícefázového podzemního proudění s názvem BOModel. K tomuto simulátoru patří ještě další programy, které s ním úzce spolupracují.

V první části se dozvíme něco o simulaci. Dále si popíšeme ložiska uhlovodíků pod zemským povrchem. Poté se dostaneme k fyzikálním veličinám popisující horniny. S tím jsou navíc spjaty jednotlivé vztahy mezi jednotlivými veličinami. Nakonec se zmíní základní výpočty a parametry simulace.

V další části si rozebereme základní princip, který se využívá v simulátoru BOModel. Seznámení se s numerickou metodou a k tomu využívaný speciální typ geometrie v geologii. Nakonec si doplníme informace o synatax vstupního souboru do simulátoru.

Následuje základní popis systému Git spolu s jeho základním použitím. Seznámení se se základními příkazy, které byly v rámci této práce využity. A zmíníme další programy spolupracující se simulátorem BOModel.

Nakonec se dostaneme k návrhu celé aplikace a k implementaci jednotlivých řešení při vývoji samotné aplikace.

Klíčová slova: C#, Git, vícefázové podzemní proudění, ložiskové inženýrství, simulátor, podzemní zásobník plynu, regulární výraz

Abstract

This thesis deals with the development of an application to help users in their work with multiphase underground flow simulator called BOModel. It consists of more programs related to the simulator that should be launched easily from this application as well.

At first, the basics of simulation are explained. The underground hydrocarbon deposits are described as well as the physical quantities of rocks. It also mentions all relations between these quantities with basic calculations.

Moreover, the main principles of BOModel are outlined. The specific numeric method that is used in BOModel is revealed with an addition of corner point geometry used in geology. The syntax of input file is explained.

The thesis further describes the basics of the use of Git system as it is used in this app including few commands. There is an additional information about programs collaborating with the BOModel simualtor.

Another part deals with the description of the application itself. There is a manual to help users work with the application. Besides, there are hints of how it has been created so if somebody is interested in continuation of the development of this application, it should provide enough information.

Keywords: C#, Git, multiphase underground flow, reservoir engineering, underground gas storage, regular expression

Obsah

Seznam obrázků

Seznam zkratek

BM64	název spustitelného souboru simulačního programu BOModel
SSH	Secure Shell
URL	Uniform Resource Locator
DLL	Dynamic-Link Library
JSON	JavaScript Object Notation
CO_2	oxid uhličitý
PZP	podzemní zásobník plynu
CVS	Concurrent Versioning System
SVN	Apache Subversion

Úvod

Cílem této práce je vytvoření spustitelné aplikace na počítače s operačním systémem Windows, která bude grafickou nadstavbou pro práci se simulátorem BOModel spolu s jeho podpůrnými programy. V podstatě se jedná o usnadnění a zefektivnění práce se vstupními a výstupními textovými soubory. O snazší správu jednotlivých projektů v čase.

Simulace jsou prováděny pomocí zadaní příkazu z příkazového řádku. Změna vstupních parametrů v souboru se mění v nějakém textovém editoru. Pro použití vizualizačních nástrojů se přepisují soubory a opět spouští přes příkazový řádek. Veškeré tyto zdlouhavé operace lze provádět snadnějším způsobem, například kliknutím na jedno tlačítko v aplikaci a o zbytek bude postaráno.

Na vyladění jednoho simulačního modelu je potřeba spousty času. Při několikahodinové nebo několikadenní práci s jedním simulačním modelem, vznikne mnoho různých verzí, které se mění v závislosti na zadaných parametrech a které je nutné nějakým způsobem si uchovávat pro pozdější porovnání. Člověk není schopen takové informace si zapamatovat, a také není vhodné využívat manuální zálohy každé změny, jež by přinášela obrovskou ztrátu času. Proto se k tomu využije automaticky řízená správa, která s pomocí jednoduché obsluhy bude dělat totéž, ale efektivněji.

1 Základy simulace

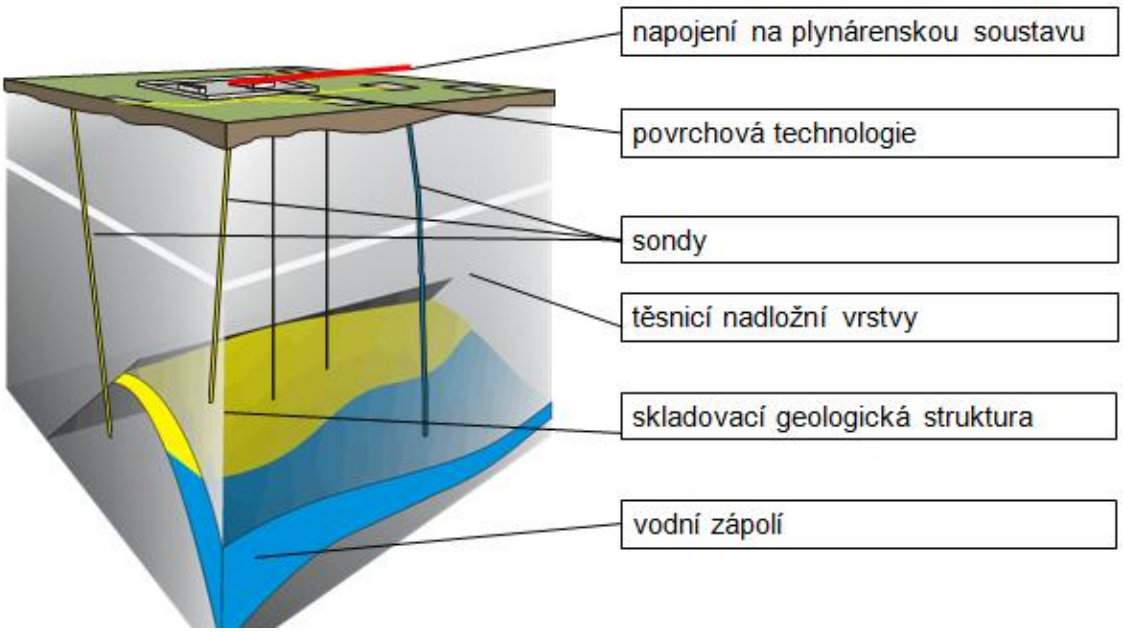
Hlavní náplní simulace je snaha o přiblížení se, s nějakým daným systémem, co nejvíce realitě. K tomu se využívá počítačového programu, tedy simulátoru, který zprostředkuje nějaké závěry na základě vstupních dat. Aby se došlo k optimálnímu výsledku pro určitý systém, ladí se simulace změnami ve vstupních datech.

Samotný program pak obsahuje komplexní matematické algoritmy k výpočtům. Simulace se neustále vylepšují a v dnešní době pronikají do všelijakých odvětví. Například se dostaly i do zábavního průmyslu do počítačových her. A na principu simulace se určuje třeba i předpověď počasí, kterou sleduje téměř každý člověk. Na tomto příkladu je i moc hezky znázorněno, že se stále jedná pouze o přiblížení se skutečnosti. Určitě si každý vzpomene, že dle předpovědi pršet nemělo, ale přeci jen zmokl.

Z těchto a z dalších mnoha důvodů se simulace zdokonalují, zpřesňují a popisují lepším a složitějším matematickým modelem, který zahrnuje více a více proměnných a parametrů, které mohou ovlivnit výsledek. Uživatel se pak pomocí úprav snaží docílit optimálního výsledku.

1.1 Ložisková simulace

Je simulace, která se zabývá ložisky, jak je uvedeno v názvu. Ložisko zde ale není míněno ve smyslu mechanicky rotačního tělesa, ale jako naleziště uhlovodíků pod zemí. Tam se nachází ložiskové pasti, které jsou zpravidla pod vrstvou těsnícího nadloží. Tato ložiska uhlovodíků mohou být jednak uměle vytvořená člověkem, nebo také přírodním způsobem. Náčrt možné podoby ložiska je na obrázku níže:



Obrázek 1 Průřez horninou s ložiskem uhlovodíku [1]

Na průřezu je žlutou barvou zvýrazněno ložisko nějakého uhlovodíku. Modrou barvou je naznačena možná přítomnost vody ze spodní strany. Nad ložiskem se nachází těsnící nadloží. Ze zemského povrchu se navrtají sondy (vrty), které jsou nějakou povrchovou technologií napojeny do nadzemního řídícího centra.

Možná ložiska tekutin mohou obsahovat následující: ropu, zemní plyn, podzemní vodu nebo CO_2. Pro simulaci, kterou se zabýváme, se seznam tekutin omezí pouze na podzemní vodu a na zemní plyn.

1.1.1 PZP

Podzemní zásobník plynu je vhodný způsob k zamezení nedostatku zemního plynu přes období topné sezóny. Z důvodu velké vzdálenosti plynového potrubí a nedostatečné rychlostí je nutné v době, kdy není takový odběr plynu, si přebytky uchovat v blízkosti, aby distribuce probíhala efektivně. Například pro Českou republiku se většina plynu těží na poloostrově Jamal, který je vzdálený tisíce kilometrů. Tuto cestu plyn urazí zhruba za 100 hodin. Jenže tak moc dopředu nelze předpovídat počasí, tedy i odběr plynu, který je na tom přímo závislý.

Tyto zásobníky na plyn se vytváří například v místě, kde se předtím nacházela ložiska ropy nebo zemního plynu, ale již jsou vytěžená. Dále pak ve zvodnělých

propustných vrstvách. V České republice tvoří tyto dvě možnosti drtivou většinu kapacity PZP. Dalšími možnostmi jsou solné a skalní kaverny.

1.1.2 Porézní materiál

Je zpravidla sedimentální hornina, která obsahuje póry. Je možné jej rozlišovat pomocí několika základních fyzikálních charakteristik.

- *Porozita*

  Je bezrozměrná fyzikální veličina, která nám udává poměr mezi objemem pórů a celkovým objemem materiálu. Může tedy nabývat hodnot z intervalu <0 ; 1>. Toto číslo je možné vyjádřit i v procentech. V našem reálném světě se ale tato hodnota pohybuje v rozmezí (0 ; 0,43). Přičemž hodnoty nad jednu třetinu jsou již velmi ojedinělé.

- *Propustnost*

  Tato veličina již má zavedenu svoji jednotku po panu Darcym. Jednotka jeden Darcy má svoji značku 1 D a její fyzikální rozměr odpovídá jednomu metru čtverečný. Propustnost se v našem reálném světě pohybuje někde mezi 5 - 500 mD.

Ačkoliv by se mohlo zdát, že propustnost a porozita jsou téměř jedno a to samé, není tomu tak. Veličiny spolu nemusí vůbec korelovat. K demonstraci je velmi vhodné se podívat na strukturu jílů. Nějaký vzorek jílu je možné si prohlédnout téměř při každém hlubším kopání do země. Jíl obsahuje velmi velké množství pórů, ale přitom má velmi nízkou propustnost. Tuto vlastnost zajišťují kapilární síly.

Podobně klamné zdání může souviset s minimální propustností, neboli s horninou, která je nepropustná. Absolutně nepropustná hornina neexistuje. Vždy bude alespoň částečně propustná. Nicméně tento pojem se používá. A to v souvislosti s okolím dané horniny. Pokud je nějaká hornina propustná a vedle ní se nachází velmi málo propustná, pak druhá jmenovaná se dá označit jako hornina nepropustná.

- *Sycení*

  Póry v hornině jsou nasycené vodou a zemním plynem. Tato veličina nám udává, jak moc je podzemní voda zemní plyn zastoupen v dané hornině. Sycení vodou označujeme S_w, a sycení plynem S_g. Index *w* značí počáteční písmenko anglického slova *water*. a to stejné je s indexem *g* pro plyn, tedy první písmeno z anglického slova *gas*. Jednotky jsou u obou případů bezrozměrné. Pro sycení v hornině platí následující rovnice:

$$S_w + S_g = 1$$

1.1.3 Výpočty simulace

V simulaci se využívá 5 dvojic parametrů. V každé dvojici je jeden z nich pro vodu a druhý je pro plyn. Dohromady je to tedy 10 parametrů, které vstupují do simulace buď jako neznámé, nebo jako konstanty. Z deseti parametrů je osm neznámých a dva z nich jsou považovány za konstanty.

Neznámé:

- *tlak* p_w, p_g
- *rychlost* $\overrightarrow{u_w}$, $\overrightarrow{u_g}$ *(filtrační)*
- *sycení* S_w, S_g
- *hustota* ρ_w, ρ_g

Konstanty:

- *teplota* T_w, T_g

Všechny neznámé jsou ve tvaru pole. Každá z nich obsahuje prostorové souřadnice a časovou složku. Například hustota vody se uvádí ve tvaru ρ_w (x, y, z, t).

Pro výše uvedených osm neznámých, je potřeba sestavit také osm rovnic. Pomocí zákonů a vztahů z knihy [2] je možné dojít k úspěšnému dopočítání všech neznámých.

- *Zákon zachování hmoty*

  pro vodu:

$$A\frac{\partial \mathrm{p_w}}{\partial \mathrm{t}} = -\nabla \cdot \overrightarrow{u_w} - \mathrm{G_w}$$

  pro plyn:

$$A\frac{\partial \mathrm{p_g}}{\partial \mathrm{t}} = -\nabla \cdot \overrightarrow{\mathrm{u_g}} - \mathrm{G_g}$$

- *Darcyho zákon*

  pro vodu:

$$\overrightarrow{u_w} = -K \cdot \nabla \cdot \mathrm{p_w}$$

  pro plyn:

$$\overrightarrow{u_g} = -K \cdot \nabla \cdot \mathrm{p_g}$$

- *Stavové rovnice*

  pro vodu:

  $\rho_w = konstanta$ nebo $\rho_w = c \cdot p_w$

  pro plyn:

$$\mathrm{p_g}\frac{1}{\rho_g} = \mathrm{R} \cdot \mathrm{T_g} \cdot \mathrm{z}$$

  ;z(p_g, T_g, chemické složení) je kompresibilitní faktor

- *Vztah pro saturace*

$$S_w + S_g = 1$$

- *Vztah pro tlaky*

$$\mathrm{p_w} = \mathrm{p_g} \cdot \mathrm{p_{kapilární}}$$

Tyto rovnice jsou buď nelineární parciální diferenciální, nelineární algebraické, nebo lineární algebraické. K vyřešení soustavy rovnic takovýchto typů nelze využít analytické metody. K vyřešení je proto nutné použít nějaké numerické metody.

1.1.4 Vstupní data

Pro vlastní simulaci ložiska jsou potřeba vstupní data, která jsou následující:

- *Geometrie oblasti, rozdělení na síť*

  Je zatížena velkou nejistotou.

- *Vlastnosti hornin*

  Stlačitelnost, porozita, propustnost. Tyto veličiny jsou v reálné simulaci zatíženy velkou nejistotou.

- *Vlastnosti tekutin*

  Hustoty, stlačitelnost, chemické složení.

- *Počáteční stav*

  Pro simulaci v čase je důležitý počátek, od něj se odvíjí zbytek simulace. Rozložení tlaků, sycení a rychlostí.

- *Okrajové podmínky*

  Definují se stavy na krajích zkoumané oblasti.

- *Scénář*

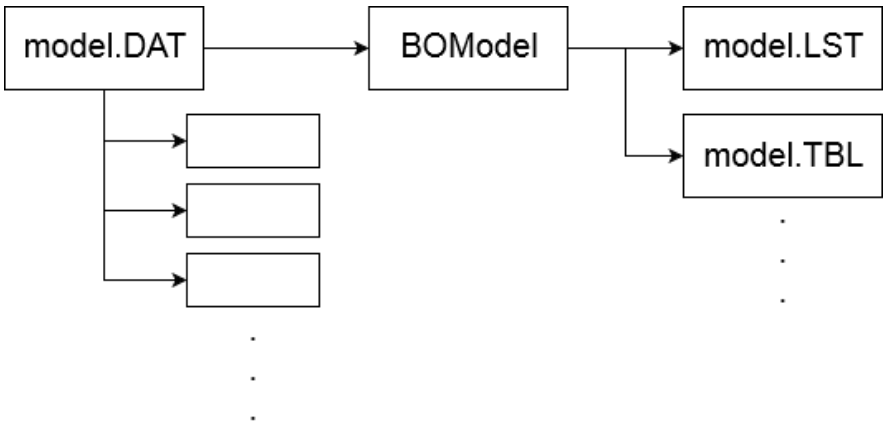
  Určuje jak moc těžíme nebo naopak jak moc vtláčíme plyn.

- *Parametry simulace*

  Určuje parametry pro řízení numerické metody, například kritéria konvergence. Určuje typ a nastavení řešiče. Jaké veličiny se vloží do výstupních souborů.

1.2 BOModel

Je simulátor, tedy počítačový program k nasimulování podzemních ložisek vody a zemního plynu. Jeho vstupem je textový soubor s příponou .DAT, který dále pracuje s dalšími soubory obsahujícími data. Tento soubor se přidá ke spuštění aplikace BOModel jako parametr. Výsledkem úspěšné simulace jsou dva textové soubory se stejným názvem, jako byl název vstupního souboru. Přípony těchto souborů jsou .TBL a .LST. Z nich je pak možné vytvořit vizualizaci dosažených výsledků pomocí dalších programů. V případě, kdy se objeví chyby v simulaci, je v souboru .TBL vypsáno chybové hlášení.



Obrázek 2 Struktura simulátoru

1.2.1 Vstupní soubor

Je textový soubor s příponou .DAT. Má svoji danou strukturu, která je složena z klíčových slov a přidaných parametrů, také má svoji syntax, čímž to trochu působí jako jednoduchý programovací nebo popisovací jazyk. Především obsahuje důležitá vstupní data pro simulaci viz 1.1.4.

1.2.1.1 Základní struktura

- *RUNSPEC*

  První částí jsou parametry simulace, které je možno nastavit.

- *GRID*

  Určuje typ geometrie, jaký typ sítě se použije, a jaké jsou vlastnosti hornin.

- *EDIT*

  Navazuje na předchozí část *GRID*. Zde se upravují parametry sítě.

- *PROPS*

  Zkratka pro anglické slovo properties. Dochází zde k nastavení vlastností tekutin.

- *REGIONS*

  V této části se v rámci oblasti definují podoblasti.

- *SOLUTION*

  Zahrnuje počáteční a okrajové podmínky. A také podmínky pro derivace.

- *SUMMARY*

  Shrnutí, které umožňuje určit, jaké všechny výstupní parametry, se budou ukládat do výstupního souboru ze simulátoru s příponou .TBL.

- *SCHEDULE*

  Jako poslední část je scénář, ve kterém se nastavuje činnost sond. Na základě těžby nebo vtláčení do ložiska.

1.2.1.2 Syntaxe

Pro zvýšení přehlednosti v zápisu je znak _ považován v následujícím výpisu za mezeru nebo za tabulátor.

- `--`

  Značí komentář do konce řádku.

- *

  Označuje defaultní hodnotu parametru.

- *číslo**

  V tomto případě *číslo* značí počet, kolik parametrů bude nastaveno na defaultní hodnotu.

- *číslo1*číslo2*

  Speciální operátor vytvořený pro BOModel. Neznačí to klasické násobení, ale vkládání čísel za sebe. Pro ukázku si ukážeme například 4*23. To vytvoří sekvenci čtyř čísel o hodnotě 23 oddělených mezi sebou mezerou, tedy: 23 23 23 23

- /

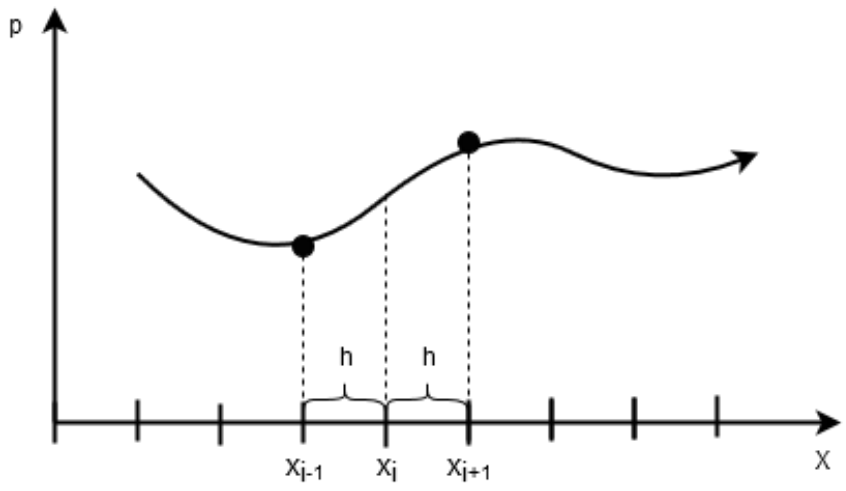
  Tento znak slouží k ukončení řady klíčových slov.

1.2.2 Numerická metoda

Metoda konečných diferencí, neboli metoda sítí, je používaná pro výpočty simulace v simulátoru BOModel. Principem je nahrazení derivací diferenčními podíly. Pro tlak by vypadalo použití střední diference následujícím způsobem:

$$\frac{dp}{dx} \sim \frac{p_{i+1} - p_{i-1}}{2 \cdot h}$$

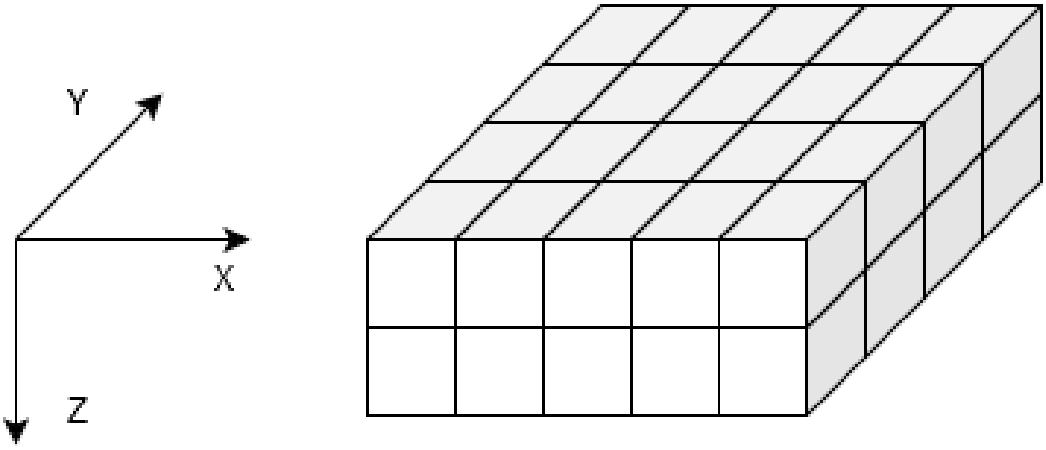
BOModel používá tuto aproximaci derivací ve všech třech prostorových souřadnicích.



Obrázek 3 Graf diferenční metody pro tlak

1.2.2.1 Definice sítě

Používá se strukturovaná síť. Jejím základem je kvádr, který se dále dělí na menší kvádry (buňky). Takovéto rozdělení je vhodné pro užití kartézského souřadného systému. V dnešní době je tento typ sítě již překonán, nicméně v našem případě se z historických důvodů stále využívá.



Obrázek 4 Strukturovaná síť rozdělena na menší kvádry

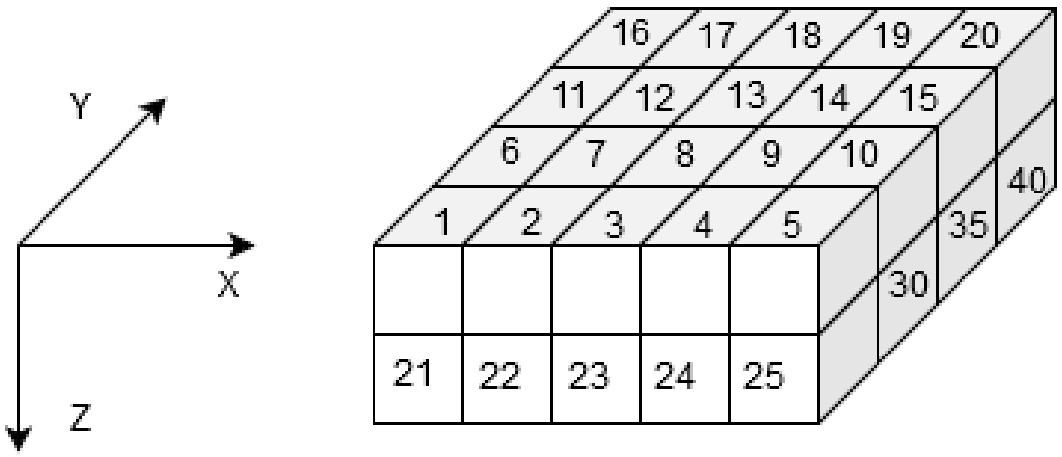
Z Obrázek 4 je vidět, že každá z os může být rozdělena na jiný počet částí. k zapsání počtu rozdělení pro jednotlivé osy souřadného systému se využívá proměnných NDIVX pro osu X, NDIVY pro osu Y a NDIVZ pro osu Z. v tomto příkladu jsou hodnoty následující:

NDIVX = 5, NDIVY = 4, NDIVZ = 2

Pokud se tato čísla spolu vynásobí, výsledkem je počet toho, kolik obsahuje velký kvádr menších kvádrů. V tomto konkrétním příkladě se dostaneme k číslu 40.

Aby bylo možné určit indexy jednotlivých kvádrů, je potřeba zvolit prioritní osu, dle které se začne indexovat. Indexování začíná od jedničky a zvyšuje se nejprve v souřadnici osy X, poté v Y a nakonec v Z.

Číslování v našem příkladu vypadá následovně:



Obrázek 5 Strukturovaná síť rozdělena na menší kvádry s indexy

1.2.2.2 Geometrie

Kvádr rozdělený na menší kvádry s indexy je nyní potřeba upravit a zdeformovat podle nějakého typu geometrie. V tomto případě se hojně využívá geometrie s názvem Corner Point Geometry, která je v oblasti podzemního těžení takovým standardem. Tato geometrie je totiž schopná vystihnout všechny tvary v běžné geologii. Postup obsahuje dva kroky.

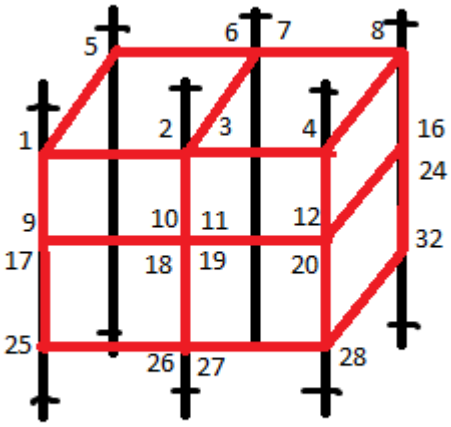
- *Definování přímek*

  Vytvoření vertikálních přímek. Počet přímek se rovná součinu $(NDIVX + 1) \cdot (NDIVY + 1)$. Na každé přímce si vyznačíme dva body, aby se z nich staly úsečky. Na těchto úsečkách pak leží vrcholy jednotlivých buněk, neboli malých kvádrů.

- *Vrcholy buněk*

  Každá buňka má osm vrcholů. Na každou úsečku vložíme vrcholy, což jsou vlastně Z-ové souřadnice.

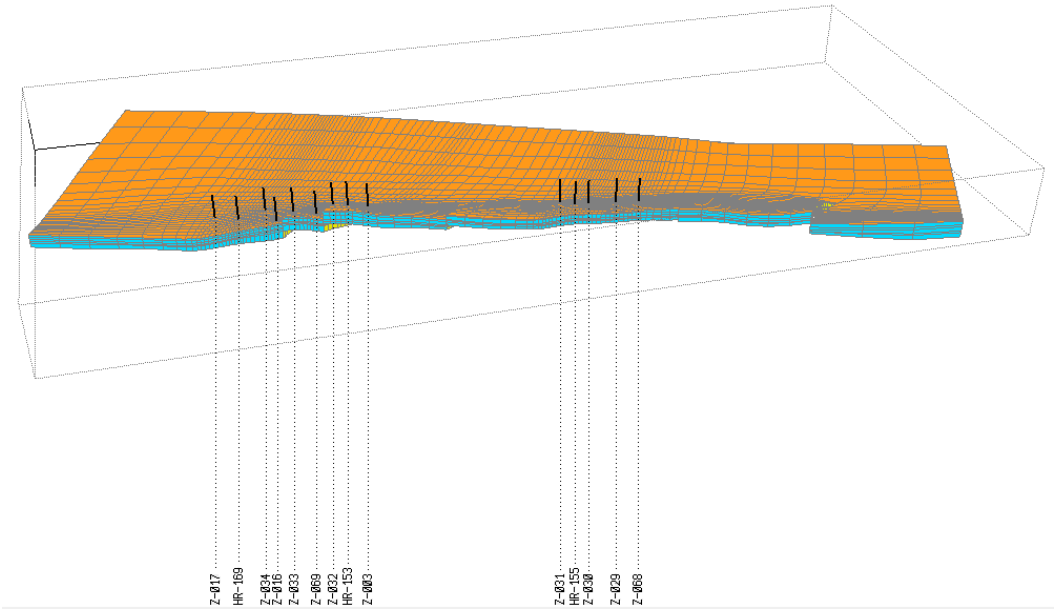
Na obrázku níže je výsledný náčrt pro NDIVX = 2, NDIVY = 1 a NDIVZ = 2.



Obrázek 6 Deformace do corner point geometry

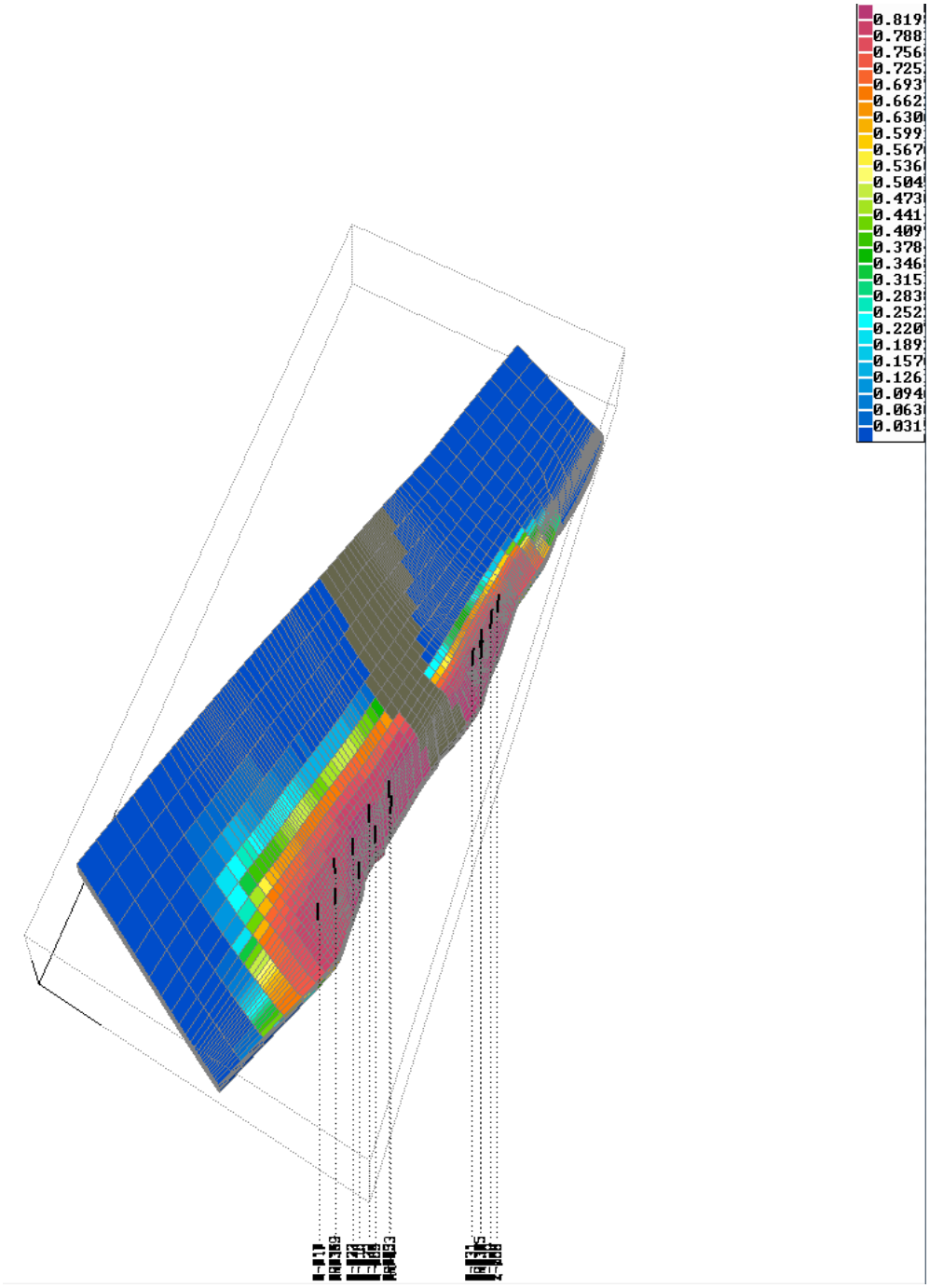
1.2.3 Reálná síť

Struktura kvádru je sice zachovaná, ale díky pružnosti použité geometrie, je možné ji libovolně zhustit nebo roztáhnout a může také dojít k trhlinám. Buňky malých velikostí bývají v blízkosti sond, kde dochází k zajímavějším jevům než v místech s buňkami velkých velikostí, kde jsou například rychlosti kapalin malé.



Obrázek 7 Model reálné sítě s vrty

S hustotou sítě přímo souvisí i sycení plynu. V hustější oblasti je sycení vysoké.



Obrázek 8 Sycení plynu v reálné síti

2 Programy

V této části práce se zaměřím na popsání důležitých programů, které bylo vhodné využít k efektivnímu výsledku. Některé věci jsou již udělány ve výborné kvalitě a nemá téměř žádnou cenu je vytvářet znovu. Jiné jsou zase již natolik složité, že by jejich vytváření zabralo o mnoho více času než celý zbytek zadané práce. A pak také existují programy, které jsou již v praxi zažité, a lidé mají práci s nimi velmi dobře osvojenou.

2.1 Git

Existuje mnoho metod ke správě souborů v čase, ale v dnešní době je jeden z nejrozšířenějších verzovací systém Git, který celosvětově využívá několik desítek milionů lidí. Jeho výhodou je jeho široká škála využití. Poměrně velmi důležitou funkcí je možnost sledování změn mezi různými verzemi sledovaných souborů a případné vrácení se v čase k nějaké starší verzi. Git lze nainstalovat na všech běžně rozšířených platformách a umožňuje lidem lehce obsluhovat i složité projekty.

Obrázek 9 Logo Git [3]

Většina operací se systémem Git je prováděna lokálně na vlastním disku vašeho počítače. Tímto přístupem se liší od téměř všech ostatních systémů. Také je to jedna z největších výhod, proč dát přednost právě Gitu před jiným verzovacím systémem. Pro základní činnosti tedy není potřeba připojení k Internetu. Na základě toho je Git velmi rychlý a neovlivňuje jej tedy ani rychlost internetového připojení. Nicméně k plnému využití tohoto systému je výhodné si jej propojit pomocí nějaké služby na vzdálené servery. S tím pak dále souvisí online služba Git Hub, která z lokálního verzovacího systému vytvoří verzovací systém propojený a přístupný po celém světě.

2.1.1 Instalace

Jak již bylo řečeno, Git je možno nainstalovat na většinu zařízení. Konkrétně na zařízení s Windows. Dále také na počítače se systémem Mac OS X. Aby mohl Git

používat opravdu každý, je k dispozici i pro Linux a jeho verze. Na všechny operační systémy je volně ke stažení z webových stránek zde. Stačí si jen vybrat správnou verzi pro vaše zařízení a nainstalovat jej běžným způsobem. V případě instalace na operační systém Windows je při samotné instalaci možné si jej nakonfigurovat dle libosti a dle vlastních preferencí, ale pokud necháte zatržené předem zvolené možnosti, nainstaluje se Git korektně. Výhodné je si zkontrolovat, zda váš počítač disponuje nejnovější verzí. V případě spuštění konzole je možné si Git aktualizovat pomocí Gitu, a to vepsáním následujícího příkazu:

- `git clone git://git.kernel.org/pub/scm/git/git.git`

Po stisku klávesy Enter se provede aktualizace, a tím budete mít zajištěné nejnovější dostupné vylepšení a opravy případných chyb.

2.1.2 Vytvoření lokálního repozitáře

Ať už je verze Gitu jakákoliv, je dobré si pomalu začít osvojovat jeho funkce. První takovou funkcí je založení vlastního repozitáře. Zpravidla se to pojí nejprve s vytvořením složky někde na vlastním disku, kde budeme chtít repozitář uchovávat. V případě, že místo je vybrané a složka otevřená, pak si v ní stačí tevřít konzoli Gitu. Nezáleží na tom zda přes příkazový řádek, nebo ve Windows přes menu na pravém tlačítku. Nyní stačí zadat jednoduchý příkaz.

- `git init`

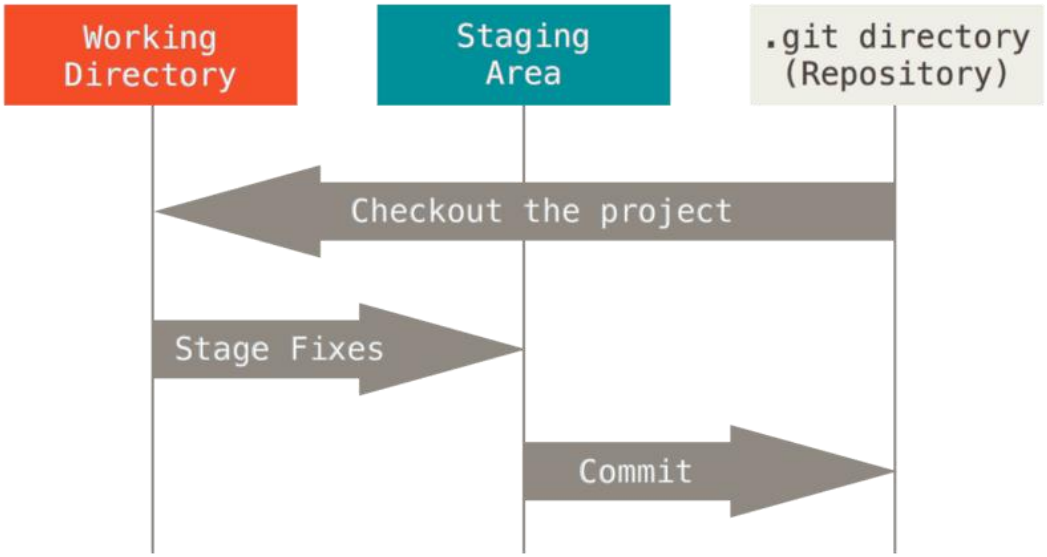
  Ve vybrané složce vytvoří repozitář. Tedy vytvoří složku *.git*, která bude obsahovat v budoucnu veškerá důležitá data. Složka je specifikovaná jako skrytá a není tak v průzkumníku viditelná.

Důležité je, že složka, ve které se rozhodneme vytvořit náš repozitář, nemusí být prázdná. Příkazem *git init* se již obsažené soubory nevymažou a vše zůstane v původním stavu.

2.1.3 Základní fáze

V Gitu se soubory dostávají do několika základních fází. v lokální verzi, tedy na Vašem počítači, se rozděluje do tzv. tří stavů [4]. Tyto stavy jsou přehledně vykresleny na obrázku níže, viz Obrázek 10. Aby se soubor dostal z jednoho stavu

do jiného, tak se v zásadě využívá různých příkazů, které se zadají do konzole nebo do nějakého, většinou grafického, softwaru třetích stran, ale toto téma bude dále rozvinuto v další části této práce.



Obrázek 10 Tři stavy souboru [5]

Jednotlivé stavy jsou nazývány jako Pracovní adresář (vlevo), Připravené změny (uprostřed) a Repozitář, který je napravo.

2.1.3.1 Git status

Příkaz *git status* je bezesporu jeden z nejpoužívanějších příkazů. Tento příkaz se hodí používat téměř neustále, protože díky němu se snadno zjistí, zda vaše operace, které zrovna provádíte, jsou v takovém stavu, v jakém je chcete mít. Přehledně vypisuje stavy jednotlivých souborů a dále umí našeptávat, jaké další příkazy v danou chvíli můžete využít a co ovlivní. Slouží tedy víceméně jako výborná kontrola, ale i jako možná nápověda v případech nejistoty. Pro uživatele začátečníky je tento příkaz naprosto nepostradatelný a jeho využití bude téměř neustálé.

2.1.3.2 Příprava k zápisu

Je proces, který dostane soubory z pracovního adresáře do stavu připravených změn. Tyto soubory musí být oproti předešlé verzi nějak upravené. Případně se jedná také o soubory nové, anebo smazané. Nicméně se nemusí ani přidávat soubory celé,

ale třeba jen určité části. Například pokud jde o jednoduchý textový soubor, pak je možné vybrat pouze některé řádky a přidat, nebo odebrat jen ty vybrané, přitom změněných bude mnohem více.

Pokud tedy chceme provést tuto operaci, využijeme příkazu *git add*. Tento příkaz má několik svých variant. Každou z nich je vhodné využít v jiném případě. Postupem času se uživatel naučí rozlišovat, kterou variantu v daný okamžik použít, aby jeho práce byla co nejvíce efektivní. Varianty příkazu jsou následovné:

- `git add—all`

  Jedna z pravděpodobně nejčastěji používaných variant. Tímto příkazem přidáme veškeré změny, co se v repozitáři udály.

- `git add -A`

  Zkrácená verze předchozího příkazu. Dělá úplně to stejné. Je důležité dodržet velké písmeno.

- `git add .`

  Tato varianta přidá veškeré změny v kořenovém adresáři. Nicméně už nepřidá změněné soubory z podadresářů.

- `git add <název-souboru> <další-název-souboru> ...`

  V případech, kdy je potřeba přidat pouze několik souborů, je tato varianta vhodná. Je možné napsat názvy daných souborů a oddělit je mezerou.

- `git add <název-souboru>`

  Opět zjednodušená verze předchozího příkazu, kdy je potřeba přidat pouze jeden soubor.

Příkazem opačným k *git add* je *git reset,* ten plní protichůdnou funkci, tedy pokud bychom například chtěli přidat všechny až na jeden, pak je vhodné přidat příkazem všechny a ten jeden nechtěný soubor odebrat.

- `git reset <název-souboru>`

  Příkaz odebere jeden vybraný soubor ze stavu připravených změn zpátky do pracovního adresáře.

- `git reset <název-souboru> <další-název> ...`

  Odebere vypsané soubory oddělené mezerou ze stavu připravených změn zpět do pracovního adresáře.

V novějších verzí je *git reset* nahrazen příkazem novějším.

- `git restore—staged <název-souboru> ...`

  Příkaz provádí stejnou operaci jako *git reset*. Tento příkaz se vám zobrazí v nápovědě při použití *git status*.

2.1.3.3 Zapsání změn

V případě, že jsme dostali veškeré chtěné soubory ze stavu pracovního adresáře do stavu připravených změn, následuje další krok. V tomto kroku dostaneme tyto soubory do dalšího stavu, tedy do stavu, kdy se nám vytvoří nová verze v repozitáři. Tento proces je opět spojován s určitou sadou příkazů. Na začátku je dobré se ujistit pomocí *git status* příkazu, zda jsou všechny soubory ve správném stavu. V kladném případě můžeme přistoupit k procesu, kterému se říká anglicky commit, tento výraz se používá celosvětově a mnoho lidí jej bude v češtině i skloňovat. Pokud se budeme snažit získat překlad tohoto slova, získáme v zásadě definici přesně toho, co chceme. My se chceme uchýlit k zapsání změn a navíc jsme odhodlaní tento krok provést. Z tohoto vysvětlení to trochu zní, že už pak není cesty zpět. Ono tomu téměř tak je, ale opět se najde šikovný příkaz, který je schopen provedené změny vrátit zpět.

Když se provádí zapsání změn, čímž se vytváří nová verze, je vhodné jí přidat nějaký popis. Ve většině případů se bude zapisovat stručný přehled změn nebo vylepšení. Je to velmi praktické pro budoucí vyhledávání v historii změn. Také přispívá ostatním uživatelům ke snazšímu pochopení vývoje. Git automaticky uloží přesné datum a čas, kdy došlo k zapsání dané změny. Navíc jej ještě opatří unikátním

klíčem, který slouží jako identifikátor dané změny. Prvním příkazem je *git commit*. A jako většina příkazů v Gitu má i tento několik svých variant:

- `git commit -m „text“`

  Nejčastější variantou je zapsání změny tímto způsobem. Do uvozovek se napíše zpráva popisující změny této verze oproti předchozím.

- `git commit`

  Tahle varianta vypadá kratší, ale ve výsledku je pomalejší, protože dle nastavení otevře určený textový editor, který jste si vybrali. Nebo otevře konzoli, kde bude chtít napsat nějaký text. Následně je třeba Gitem otevřený soubor uložit a editor zavřít. Tohle všechno se musí udělat, aby se došlo k úplně stejnému výsledku jako v předchozím případě.

- `git commit -a -m „text“`

  Nejzajímavější varianta příkazu, která propojuje příkaz *git commit -m „text“* z této fáze a příkaz *git add —all* z fáze přípravy k zápisu. Čili pomocí jednoho příkazu je možné veškeré soubory se změnami nahrát přímo jako novou verzi do repozitáře.

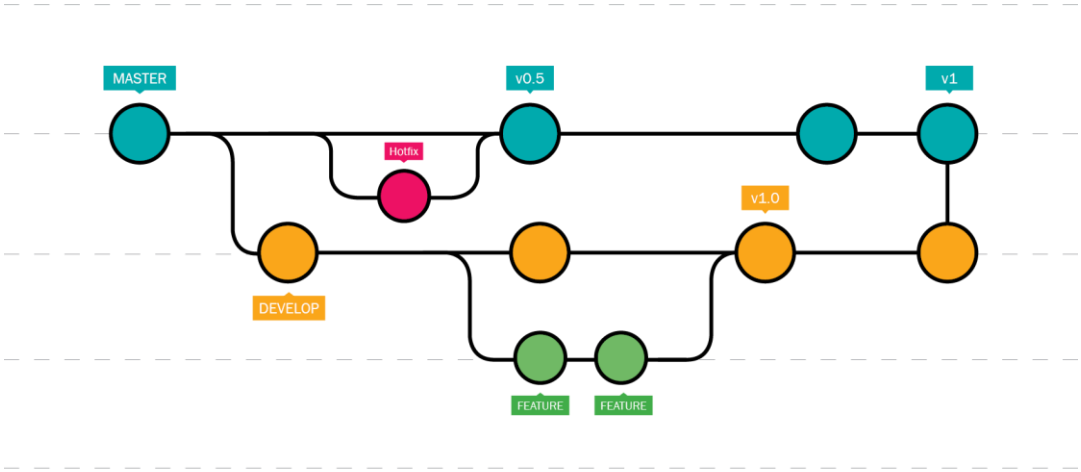
Nyní se podíváme na příkaz, který tyto změny může navrátit.

- `git revert <commit_hash(ID)>`

2.1.4 Větvení

Po vytvoření repozitáře se automaticky vytvoří i hlavní větev, která je nazvaná jako *master*. Je to vlastně jedna linie, která se prodlužuje přibývajícími verzemi. Nicméně občas se může stát, že není jen jedna cesta vpřed, ale hned několik. Na počátku není jasné, která bude tou správnou cestou, a tak se vývoj rozdělí do několika paralelních větví. Hlavní větev zůstává a k tomu se například přidají dvě další. Tato situace se dá poměrně dobře představit jako veliké vlakové nádraží, kde je kolej hlavní a pár vedlejších, pak také koleje odstavné. V zásadě může nejčastěji dojít ke dvěma koncům dané větve. Buď se postupem času zjistí, že toto řešení nikam nevede a první větev skončí jako slepá, nebo se dojde k závěru, že to co se

v té druhé vytvořilo je přínosné a sloučí se s větví hlavní. Pro úplnost představy je uveden názorný obrázek jednoduchého větvení v Gitu, viz Obrázek 11.



Obrázek 11 Příklad možného větvení v Gitu [6]

Modře je na obrázku vyvedena hlavní větev *master*. Růžová větev značí pouze rychlou opravu, kde došlo ihned k sloučení do větve *master*. Oranžová větev je vývojová. V ní se věci vyvíjí, a když jsou hotové, sloučí se do hlavní větve. Dále je ale možné i z vývojové větve oddělit další větev (zelenou), ve které se zkouší další nové nápady. Ty se v úspěšném případě spojí zpět do vývojové větve, ve které se dále pokračuje již s vytvořenými nápady. Pokud by nápady dopadly špatně a nepodařilo se je vytvořit, pak by větev zůstala jako slepá a nespojila se s oranžovou větví.

2.1.4.1 Git branch

Tento příkaz funguje v obdobném smyslu jako *git status*. Nicméně nyní se jedná pouze o výpis všech větví, které jsou vytvořeny v daném repozitáři. Pokud ještě nedošlo k vytvoření nové větve, tak výsledek po tomto příkazu bude pouze větev *master*, která existuje vždy. Dále se může příkaz hodit k využití, pokud již existuje mnoho větví a člověk si nepamatuje, do které z nich chce přejít. Také je při použití tohoto příkazu zvýrazněna větev, která je v daném okamžiku aktivní. Tato skutečnost se pozná dle symbolu hvězdičky u jejího názvu.

2.1.4.2 Založení nové větve

Když se v určitém momentu rozhodneme, že je vhodný čas k vytvoření nové větve, pak je důležité si ji dobře pojmenovat. Příkaz k vytvoření nové větve je velmi jednoduchý a vypadá následovně:

- `git branch název-nové-větve`

  Název nové větve by měl být bez mezer, a také by neměl být příliš dlouhý kvůli následnému pohodlí při přepínání mezi větvemi.

Po založení nové větve je dobré si zkontrolovat pomocí příkazu *git branch*, zda byla větev správně vytvořena.

2.1.4.3 Přepínání mezi větvemi

Přepnutí z jedné větve na druhou v rámci jednoho repozitáře není tak složité. Nicméně je potřeba dodržet pár zásad. Nelze přepínat mezi větvemi, pokud v aktuální větvi není vše dokončené. Pokud ale nejsou nalezeny žádné změny a vše je nahráno v repozitáři, pak je možné za pomoci níže uvedeného příkazu přejít do jiné větve.

- `git checkout název-větve`

  Příkazem dojde k přesunutí do poslední verze v dané větvi definované jejím názvem.

- `git checkout -f název-větve`

  Tato modifikace dovoluje násilné přepnutí do jiné větve.

Pokud je potřeba se přepnout zpět, tak se opět musí využít stejného příkazu za stejných podmínek. Potřebujeme-li se dostat do hlavní větve, pak použijeme příkaz *git checkout master*.

Existuje příkaz, který dokáže sjednotit příkazy *git branch* a *git checkout*. Tím se ušetří nějaký čas a několik úhozů do klávesnice.

- `git checkout -b název-nové-větve`

  Provede se vytvoření nové větve s určeným pojmenováním, a dojde rovnou i k přepnutí do nové větve

Git vám sám vypíše, že došlo k přepnutí do jiné větve včetně jejího názvu. Tím je provedena i kontrola, tudíž není nutné využít příkazu *git branch*.

2.1.4.4 Sloučení větví

Jakmile je jednou dokončena práce ve vytvořené větvi, bude potřeba nějak dostat výsledek zpátky do hlavní větve, případně i do jiné. Opět to není nijak složité. Důležité je se nejprve přepnout do větve, do které chceme slučovat. Pokud se rozhodneme slučovat do hlavní větve (master), pak to provedeme příkazem *git checkout master*. V tuto chvíli bude vše připraveno na použití následujícího příkazu:

- `git merge název-větve`

  Pomocí tohoto příkazu se sloučí větev *název-větve* s větví, která je momentálně aktivní.

2.1.4.5 Vymazání větve

Pokud se do sebe sloučí dvě větve, pak je jedna z nich nadále zbytečná a je možné ji smazat, protože je již obsažena v té druhé a nemá smysl je obě udržovat. Ke smazání větve nicméně mohou vést i jiné důvody a jedním příkazem je hotovo.

- `git branch -d název-větve`

  Využijeme předchozího příkazu *git branch* a přidáme *-d*, které značí anglické „delete“, tedy smazat.

Je důležité být přepnutý v jiné větvi než v té, která se bude mazat. Git nedovoluje smazat aktivní větve. V případě, že jste ve větvi, kterou chcete smazat, je nutné se příkazem *git checkout* přepnout do jiné větve, a teprve poté ji vymazat.

2.1.5 Git Hub

Na webových stránkách Git Hub si uživatel založí svůj účet, a tím se mu otevře možnost si vytvořit vlastní repozitáře (projekty), nebo si stáhnout repozitáře jiného

uživatele. Dalo by se to téměř nazvat cloudovým úložištěm lokálních repozitářů, kam si uživatel nahraje, co potřebuje. Díky této službě má přístup ke svým datům odkudkoliv. Své repozitáře je možno mít nastavené buď jako soukromé, nebo jako veřejné. Záleží jen na uživateli, kterou možnost si vybere.

2.1.5.1 Konfigurace uživatele

Po založení účtu je vhodnou volbou uložit si své údaje i do Gitu na svém počítači. Nejjednodušší varianta je autorizace pro veškeré repozitáře. Po zapnutí konzole stačí využít dvou příkazů, a dále nebude nutné se o tuto záležitost starat.

- *git config—global user.name „vaše-přihlašovací-jméno"*

  Do uvozovek se vpíše uživatelské jméno, které si uživatel vytvořil na serveru Git Hub.

- *git config—global user.email „váš-e-mail"*

  Opět se do uvozovek napíše e-mail, který je přiřazený ke stejnému účtu na Git Hub.

2.1.5.2 Vytvoření repozitáře

Při vytváření nového repozitáře na Git Hub je důležité jeho jméno, které by mělo být především krátké a výstižné. Aby jen název neurčil, co se v repozitáři bude skrývat, je možné k němu dopsat popis. Opět by měl být stručný a přehledný. Nicméně je dobrovolný, a proto se nemusí napsat vůbec žádný. Více informací lze ideálně napsat do textového souboru *README*, jenž je možné přidat do repozitáře už při jeho založení.

Dále lze přidat soubor *.gitignore*, který obstarává, že se určité typy souborů, složky nebo i jen jednotlivé soubory nedostanou do správy Gitu. Tento soubor lze přidat i později a není to žádný problém.

Nadále je možnost výběru licence, pod kterou se může repozitář a soubory v něm šířit. Na výběr jich je velké množství.

Pokud se vytváří repozitář na Git Hub pro již existující lokální repozitář, pak se s možnostmi výběru licence se souborem *.gitignore* a s *README* nezabýváme a přeskočíme je.

Nakonec se tvůrce repozitáře musí rozhodnout, zda svůj repozitář nastaví jako soukromý, nebo jako veřejný. Vybraná možnost se dá kdykoliv později změnit.

Soukromý

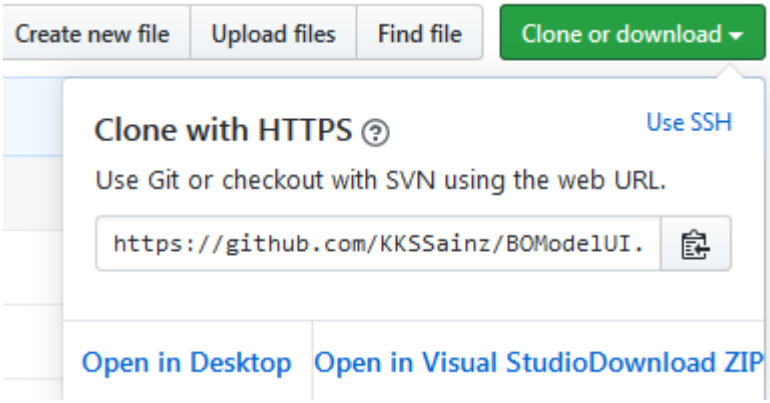
Repozitář s tímto omezením je viditelný pouze pro uživatele, který ho vytvořil. Ostatní uživatelé neuvidí ani, že existuje. Nicméně je možnost v nastavení přidat ručně dalšího uživatele, který daný repozitář uvidí a který s ním bude moct dle libosti pracovat.

Veřejný

Takto nastavený repozitář je vidět v seznamu uživatele, který jej vytvořil. Je možné ho najít dle názvu pomocí vyhledávání na Git Hub. Tudíž kterýkoli uživatel má k takovémuto repozitáři přístup, může si ho prohlížet, stáhnout k sobě na lokální disk, a také nahrávat zpět změny, které musí být ale schváleny.

2.1.5.3 Stažení repozitáře

Vybraný repozitář je možné stáhnout z Git Hub na svůj lokální disk. Je na výběr stažení buď pomocí SSH klíče nebo skrze URL odkaz. Aktuální provedení je k vidění na Obrázek 12.



Obrázek 12 Možnosti stažení repozitáře

Po zkopírování si adresy do schránky, nalezení složky na lokálním disku, kam se má repozitář nahrát, je potřeba spustit konzoli a vepsat příkaz:

- *git clone URL(SSH)*

  Ve vybraném místě na disku, vytvoří klon repozitáře z Git Hub, který je určen pomocí odkazu nebo klíče.

Poslední verze

Stažení posledních změn ze vzdáleného repozitáře je možné pomocí dvou příkazů, které se od sebe trochu liší:

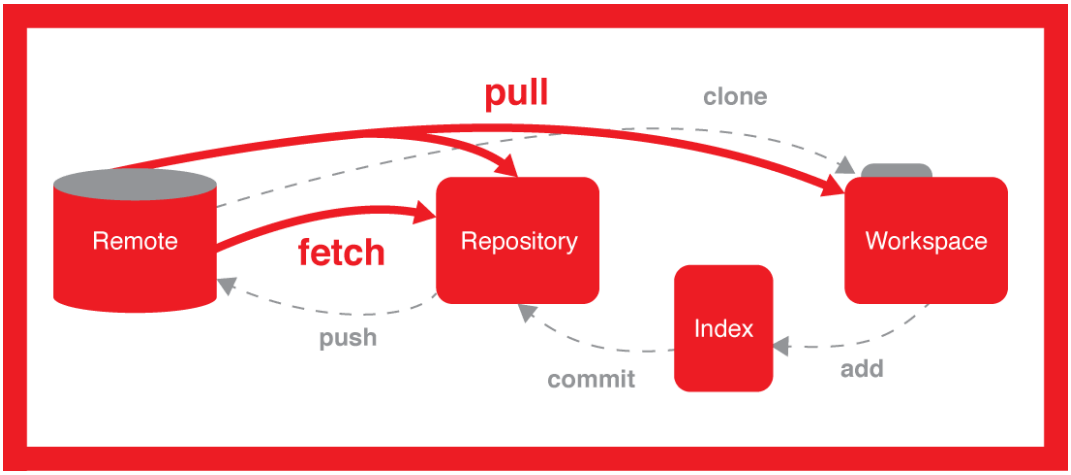
- *git fetch*

  Nahraje změny ze vzdáleného repozitáře do lokálního a vyvede je jako novou větev. Tedy dojde ke stažení nejnovějších dat, ale dál s nimi nepracuje.

- *git pull*

  Tento příkaz zvládne více věcí naráz. Stáhne a aktualizuje data ze vzdáleného repozitáře, tak jako *git fetch*. Navíc ještě dojde k přepsání samotných dat v pracovním adresáři na lokálním disku.

Názorný přehled obou příkazů je možné si prohlédnout na přiloženém obrázku, viz Obrázek 13. Spolu i s naznačenými dalšími příkazy, které již byly zmíněny, anebo teprve budou.



Obrázek 13 Příkazy mezi vzdáleným repozitářem, lokálním repozitářem a pracovním adresářem [7]

2.1.5.4 Nahrání změn

Neméně důležitá část souvisí i s opačným směrem. Je potřeba nejen si poslední změny ze vzdáleného repozitáře stáhnout, ale také je tam nahrát, jinak by nebylo co stahovat. k této činnosti slouží příkaz, který obrazně doplňuje sérii od *git add* po *git commit*. Dalším krokem tedy je:

- `git push`

  Nahraje veškeré změny z lokálního repozitáře na vzdálený repozitář.

Tímto jsme uzavřeli asi základní přehled různých povelů a funkcí, které je možné s Gitem, potažmo s Git Hub-em vytvářet. Téměř všechny z nich budou v této práci využity. Buď už v samotném uživatelském rozhraní spouštěny automaticky, nebo manuálně při jejím vytváření.

2.1.5.5 Issues

Nachází se v repozitáři jako jedna ze záložek. Při práci v týmech nebo ve skupinách na nějakém projektu, je možné přiřadit individuálnímu člověku nebo více lidem určitý úkol, kterým by se měl(i) zabývat. Tyto úkoly je možné sledovat, a také se může měnit jejich priorita, nebo jejich aktuální stav.

2.1.5.6 Requests

Jsou případy, kdy si člověk s něčím neví rady a potřebuje poradit. Tím se zabývá další ze záložek na Git Hub. Jednoduše se vytvoří žádost, s čím by bylo potřeba pomoci, která se zobrazí ostatním uživatelům, a ti budou moci k vyřešení problému přispět.

2.1.6 Zadávání příkazů

Existuje mnoho způsobů, jak pracovat s Gitem. Základem bývá konzole, která je obdobná jako příkazový řádek ve Windows. Jinou možností je využít aplikací třetích stran, které jsou vyvedeny v grafickém provedení.

2.1.6.1 Konzole

Veškeré příkazy uvedené v této práci výše, ohledně práce s Gitem, jsou určené pro práci s konzolí. Je to nejsnazší způsob, jak se dá předejít omylům a nedorozuměním, například s překlady do jiných jazyků.

2.1.6.2 Grafická rozhraní

Výhodou proč používat aplikace vytvořené pro správu Gitu může být snadnější vizualizace. Nicméně se občas může stát, že každá aplikace si může interpretovat vaše pokyny trochu jinak a nemusí tedy být zaručen očekávaný výsledek. Samozřejmě, pokud si člověk na nějaký program zvykne a bude jej využívat pořád, pak s tím mít problém nebude. Existuje řada těchto aplikací, ať už se jedná o doplňky v různých vývojových prostředích, anebo samostatné programy, jako jsou například Fork, Git Cola, SourceTree, GitHub Desktop a mnoho dalších. Některé z nich jsou zadarmo, jiné je potřeba zakoupit. Ve výsledku záleží na každém, co mu bude nejlépe vyhovovat.

2.2 Knihovny

Zde bude uveden základní přehled knihoven využitých při programování uživatelského rozhraní. Ve vývojovém prostředí Visual Studio se různá rozšíření a knihovny nacházejí jako balíčky NuGet. V nich se dá snadno vyhledávat a nainstalovat pouze ty, které je potřeba. Automaticky jsou inicializovány do vytvářeného projektu.

Knihovny používané systémem Windows mají obvykle k názvu připojenu příponu *.dll*.

2.2.1 Json

Jednou z knihoven, kterou je rozhodně vhodnější stáhnout, než ji vytvářet, je knihovna *Newtonsoft.Json*. Tato knihovna dokáže například z vytvořené třídy vytvořit soubor, který má jednoduchou strukturu. Lze jej jednoduše číst, a třídit v něm uložená data. Takový soubor disponuje příponou *.json*.

2.3 Vizualizační programy

K zobrazení výsledků je potřeba nějaká grafická interpretace. Vzhledem k dlouhé historii využívání simulátoru BOModel, jsou k němu dotvořeny i programy, které zobrazí výsledky simulace. Tvůrce všech těchto programů je stejný, tudíž je zaručena i kompatibilita mezi nimi. Z důvodu, že uživatelé jsou na tyto programy navyklí a práce s nimi není nepohodlná, není cílem vytvářet jiné, které by plnily stejnou funkci.

2.3.1 Grfext

Skript je volán pomocí příkazu z příkazového řádku s parametry, které vycházejí z konfiguračního souboru určujícího přesná data k zobrazení. Výstup je nadále použit dalším programem.

2.3.2 Splint

Jako parametr se předává výsledek z předchozího programu Grfext. Tento program je již samotným vizualizátorem vybraných dat. Obsahuje mnoho nástrojů, které usnadňují práci se samotnými průběhy.

3 Aplikace

Úkolem je vytvoření jednoduše ovladatelné desktopové aplikace pro operační systém Windows s běžně známými a zaběhlými uživatelskými prvky pro snadné ovládání simulátoru BOModel a všech jeho peripetií, se kterými dále spolupracuje.

3.1 Specifikace

Pro aplikaci je důležité, aby mohla korektně pracovat, mít ve správných adresářích příslušné programy, které bude ke svému fungování využívat. Po spuštění aplikace je potřeba udělat analýzu těchto programů a zhodnotit jejich úplnost. Na základě toho by měla aplikace zobrazit, zda jí něco chybí, anebo je kompletní a připravena k plnému použití.

Aplikace dále musí umět nahrát vstupní projekty k simulaci. U nich by měl být patrný název vstupního souboru spolu s názvy doplňujících souborů. Projekty je potřeba upravovat, a tyto změny si nějak ukládat, aby se k nim dalo i po delší době vrátit. K tomu se využije verzovací systém, aby nedocházelo ke zbytečnému duplikování a k zahlcení místa na lokálním disku.

U vstupů do simulátoru panuje značná nejistota. Takových vstupů pro jeden reálný zásobník jsou klidně i desetitísíce. Ať už se jedná o parametry tekutin nebo hornin. Z ekonomických důvodů jsou tedy hodnoty na počátku odhadnuté, protože získat vzorek, který je hluboko pod povrchem, je velmi drahá záležitost. Důležitá je v tomto případě historie zásobníku. Čím bude delší a obsáhlejší, tím bude přesnější počáteční nastavení parametrů. Toho je možné dosáhnout inverzní úlohou, tedy ze známých výstupních parametrů se snažíme dostat vstupy (parametry hornin, počáteční podmínky,..). Taková úloha má nekonečně mnoho řešení, ale jednotlivé proměnné mají určitá fyzikální omezení, která je nutné dodržet. Nejprve v tzv. fázi „history matching“ se provádí simulace minulosti. V ní se snažíme dosáhnout shody tlaků a shody množství plynu v pzp. Nicméně k řešení inverzní úlohy se ve výsledku využívá ručního ladění. Samotná simulace trvá jednotky až desítky minut.

K zobrazení a k porovnání výsledků simulace pro konkrétní projekt, je třeba mít možnost k němu dodat reálná data v tabulkovém formátu. Dále je důležité mít možnost provést porovnání i s předchozími verzemi daného projektu.

3.2 Návrh

Aby bylo možné vytvořit aplikaci, je potřeba si udělat předem přehled o důležitých prvcích, na kterých bude stát její základ. Do toho se počítá především programovací prostředí s programovacím jazykem, struktura programu a jeho třídy. Ale také je potřeba brát v potaz, k čemu aplikace má sloužit a kvůli čemu vzniká.

Motivací k vytvoření aplikace se dostáváme při práci se simulátorem a při lazení parametrů modelu. Když se změní parametr, je potřeba tuto změnu porovnat s předchozí podobou nebo se snažit dostat k nějaké dané podobě. Taková věc obvykle zabere mnoho času a také se může stát, že se nejlepší průběh již uskutečnil s nějakým parametrem třeba o hodinu dříve. Za tu dobu je málo pravděpodobné, že by si člověk pamatoval přesné nastavení, tudíž by potřeboval se do daného místa navrátit. K tomu je potřeba využít automatického systému, který bude sám ukládat změny, ke kterým bude možné se snadno vracet. S tím je spjato i používání příkazového řádku ke spuštění simulace nebo ke spuštění vizualizačních prostředků. Vše je nutné ručně přepisovat a spouštět. Toto je další motivace, aby došlo k výraznému zjednodušení použití a k automatickému spouštění jednotlivých komponent.

Verzovacích systémů je možné použít rovnou několik. Jednou z možností je vytvořit si nějaký vlastní. Možností použití vytvořených systémů je také několik. Prvním z nich je systém CVS, který byl vydán v roce 1990. Používá architektury klient-server, a je to velmi dobrý nástroj pro správu a porovnání jednotlivých verzí. Dalším je systém SVN vytvořený v roce 2000. Byl navržen tak, aby byl nástupcem CVS. Jeho zásadní nevýhodou je, že pracuje pouze online. V roce 2005 byl vytvořen systém Git, který se postupně dostal na vrchol. Je nejrozšířenější, optimalizovaný. v roce 2014 se stal nejpoužívanějším verzovacím systémem na světě, kdy přeskočil systém SVN [8]. Dnes je to tedy již naprostý standard v této oblasti, a proto bude využit i v této aplikaci.

3.2.1 Programovací prostředí

Vybral jsem Visual Studio nejnovější verze, které je možné získat v určité verzi na Windows zdarma. Je zde mnoho doplňků, které mohou zrychlit nebo alespoň zpříjemnit práci při vytváření projektu. Od možnosti výběru barvy pozadí, přes

zvýraznění syntaxe, až po kompletní práci se službou Git. Dále je zde kvalitní dokumentace od Microsoftu. V tomto prostředí jsem navíc pracoval v minulých letech, a ačkoliv jsem využíval starší verze, příliš se od nových neliší. Velkou výhodou je podpora všemožných programovacích jazyků.

3.2.2 Programovací jazyk

Z počátku to vypadalo, že využiji obrovský potenciál moderního jazyka Python, ve kterém je možné vytvořit téměř cokoliv. Nicméně po několika zkušebních pracích v něm jsem došel k závěru, že bude snazší od tohoto nápadu upustit, a k vytvoření přehledné aplikace, spustitelné ve Windows, se uchýlit k balíčku .NET. A tedy i k rozsáhlému programovacímu jazyku C#. Z tohoto jazyka mám z předešlého studia na vysoké škole utuženy potřebné základy, které je možno dále rozvíjet a zlepšovat. Věřím, že volba objektově orientovaného jazyka se ve výsledku ukáže jako vhodná. V programovacím jazyku C# programuje obrovské množství lidí po celém světě, a díky tomu je možné najít rady a nápady k velké spoustě problémů, které se při vytváření mohou vyskytnout.

3.2.3 Struktura

Program je složen z hlavní části, s tím je spjat příslušný formulář, a několika tříd k tomu.

3.2.3.1 Hlavní

Součástí je grafický formulář s jednotlivými ovládacími prvky, kterým se mění vlastnosti nebo specifikují různé události. Na základě rozmístění prvků a očekávání uživatelů, je potřeba dbát na uživatelský komfort při práci s aplikací a používat zavedené zvyklosti z již osvědčených aplikací, které se používají každý den.

3.2.3.2 Třídy

Jednotlivé instance tříd jsou volány pomocí konstruktoru z hlavní části programu. Aplikace obsahuje tři třídy.

- *Prj*

  Nejdůležitější a nejobsáhlejší třída, ve které se nachází veškeré potřebné informace o jednom projektu. Její konstruktor obsahuje dvě přetížení

v závislosti na tom, zda obsahuje soubor s realitou. Dále obsahuje mnoho metod, které vrací různé podoby s cestami do daného projektu, případně názvy obsažených souborů. a upravená metoda *ToString()*.

- `Project_Json`

  Využívá vlastní doinstalované knihovny a jednotlivé instance *Prj* k vytvoření textového *.json* souboru s požadovanými proměnnými.

- `Git_Commit`

  Slouží k shromáždění důležitých informací ohledně jednoho commitu v repozitáři. Tyto informace dokáže potom v požadovaném tvaru vypsat díky metodě k výpisu.

3.3 Dokumentace

V následující části se budeme zabývat aplikací a jejím použitím. Také si rozebereme jednotlivé metody programu a jeho algoritmy.

3.3.1 Uživatelská

Základní popis postupu k používání uživatelského rozhraní pro simulátor BOModel, která se jmenuje *BOModelUI*. Návod bude konstruován a sepsán tak, aby se povedlo zprovoznit aplikaci každému uživateli na svém osobním počítači s operačním systémem Windows.

3.3.1.1 Před spuštěním

Než bude aplikaci možno vůbec získat, natož ji používat, je velmi důležité mít na daném počítači nainstalovaný Git. Popis k jeho získání a k jeho instalaci je k nalezení v části 2.1.1. Tento krok je zásadní, protože na něm závisí funkčnost celé aplikace.

Aplikaci, která je obsažena v repozitáři, je potřeba dostat na lokální disk, na místo kde ji chcete mít, a odkud se bude spouštět. Toto se provede pomocí příkazu:

- `git clone https://github.com/KKSSainz/BOModelUI.git`

Samotný repozitář se poté nachází na serveru Git Hub, kde je volně ke stažení na odkaze:

- *https://github.com/KKSSainz/BOModelUI*

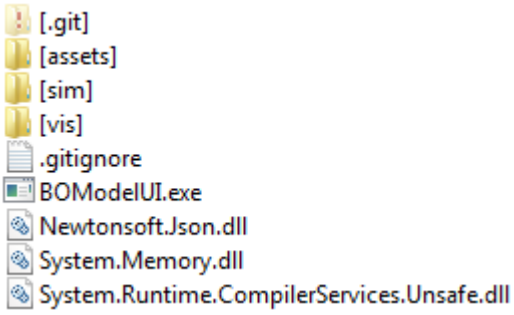
Po stažení tohoto repozitáře je potřeba v jeho hlavní složce, kde se nachází knihovny, složky vis a sim, vložit do tohoto místa repozitář assets, ve kterém se budou nacházet projekty. Odkaz na repozitář je zde:

- *https://github.com/KKSSainz/assets*

A vložení probíhá stejným způsobem jako v předchozím případě, akorát s odlišným URL.

- `git clone https://github.com/KKSSainz/assets.git`

Pokud je po vložení druhého repozitáře stav následovný, jako tomu je na Obrázek 14, pak by měla být aplikace připravena k plnému použití. Pokud by něco z důležitých věcí chybělo, aplikace sama vypíše, co ji schází, danou věc bude nutné doplnit, do té doby nebude aplikace spuštěna.

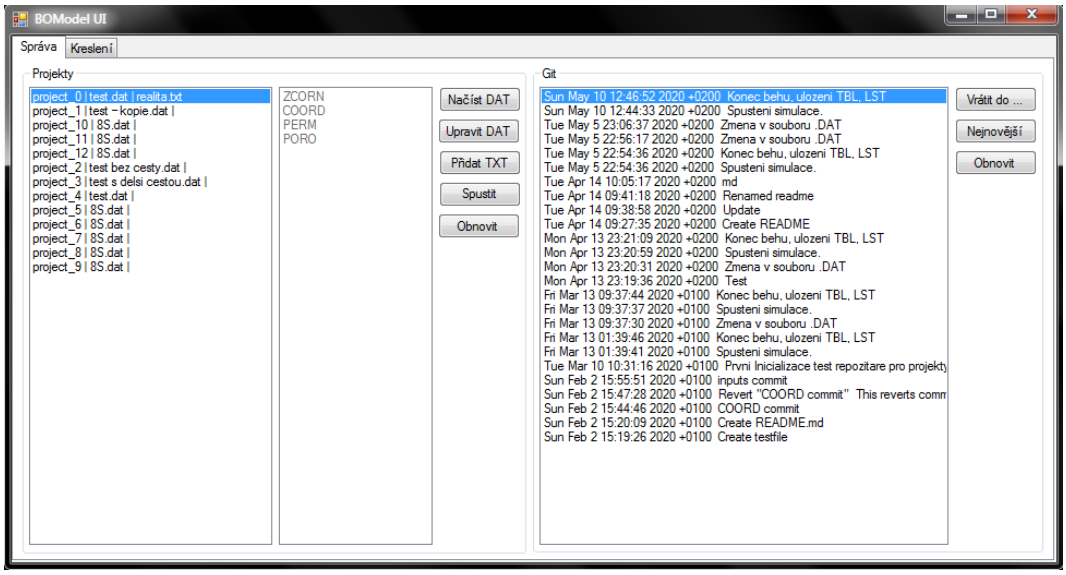


Obrázek 14 Přehled složek a souborů aplikace BOModelUI

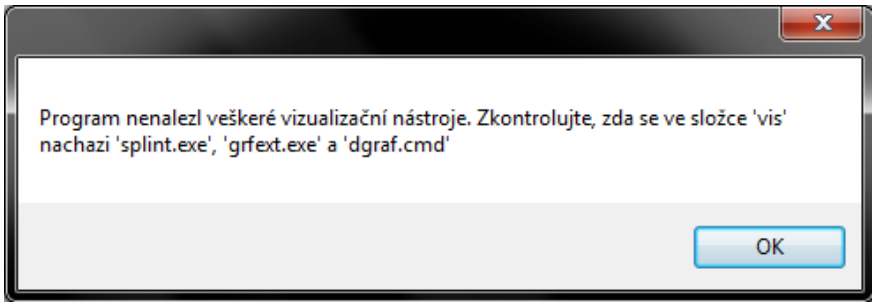
Nyní by nemělo nic bránit ke spuštění vlastní aplikace, tedy spuštění spustitelného souboru *BOModelUI.exe*.

3.3.1.2 Spuštění

Při samotném spouštění dojde ke kontrole, a pokud je vše v pořádku, aplikace spustí klasické windows okno. Aplikace je rozložena ve dvou záložkách mezi správu a vizualizaci. V opačném případě vyskočí chybová hláška s popsanou chybou, kterou je potřeba odstranit.



Obrázek 15 Aplikace po úspěšném spuštění



Obrázek 16 Chyba při spouštění aplikace

3.3.1.3 Správa

Po zapnutí aplikace je vždy aktivní záložka správy. Ta je dále rozdělena na dvě sekce. Na sekci s projekty a na sekci k ovládání verzí v Gitu.

U sekce s projekty je pole, kde se zobrazují aktuální projekty. K výběru projektu k zobrazení informací a k další práci s ním je potřeba kliknout na text s požadovaným projektem. Tím se v poli vedle ukáží další soubory, se kterými se v daném projektu pracuje. Pokud je nějaký z těchto souborů označen modře, znamená to, že chybí.

Vedle těchto polí se nachází tlačítka. První dvě slouží pro vstupní soubory *.DAT*. První z nich je přidání nového vstupního souboru, čímž se založí nový projekt. K výběru souboru je využito klasického průzkumníku, který je použit ve všech aplikacích. Druhé tlačítko je pro editaci vstupního souboru. Pro označený projekt se otevře nové okno, kde je možné jej upravit dle libosti. Po uložení a zavření

editovacího okna se v případě provedení nějaké změny program zeptá, zda chcete spustit simulaci. To je v dialogovém okně možné potvrdit a simulaci spustit okamžité, nebo ji je možné odmítnout. V případně zamítnutí se opět vrátí do výchozího vzhledu.

Tlačítko k přidání textového souboru s reálně naměřenými daty přidá jeden soubor do vybraného projektu. Z aplikace není možné jej editovat a slouží dále až k vizualizaci výsledků.

Další tlačítko je pro spuštění simulace v simulátoru BOModel. Vybraný projekt se nasimuluje. Pokud by se v simulaci vyskytla chyba, program vypíše chybové hlášení, které se nachází ve výstupním souboru *.TBL*.

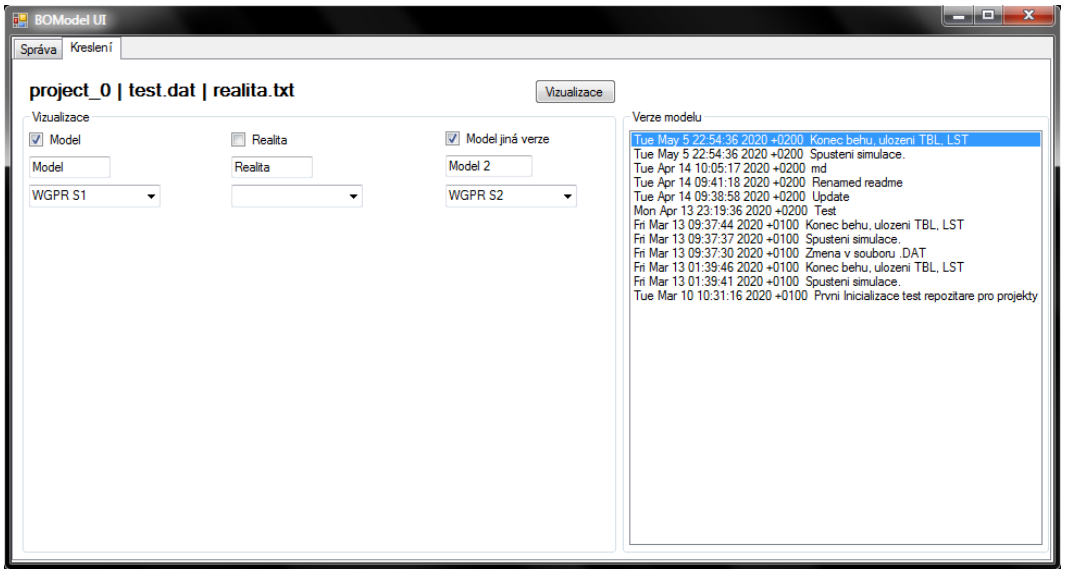
Posledním tlačítkem je možné zaktualizovat seznam projektů se seznamem jejich dalších souborů.

V sekci Git je opět pole, kde se vypisuje log, tedy aktivita v čase. Jsou zde evidovány spuštění simulace, dokončení simulace nebo editace vstupního souboru. Vždy je obsaženo přesné datum a čas, kdy byla daná událost provedena spolu s popisem.

Stejně jako u projektů je možné si jednotlivé události označit. Při označení a stisknutí prvního tlačítka je stav projektů uveden přesně do daného okamžiku v minulosti. Další tlačítko slouží k dostání se zpět k nejnovější verzi. A opět obnovovací tlačítko, které zaktualizuje seznam událostí.

3.3.1.4 Vizualizace

V levém horním rohu je možné se přepnout na druhou záložku. Nicméně to má jednu podmínku a to, že je potřeba mít v daném projektu již provedenou simulaci. Bez ní nelze do druhé záložky přejít a vyskočí chybová hláška o nenalezení souboru s příponou *.TBL*.



Obrázek 17 Přehled aplikace v záložce kreslení

Vzhled záložky je obdobný tomu, co se nachází v záložce druhé. Na pravé straně aplikace se nachází okénko s logem z repozitáře. Tentokrát ale jsou zde uvedeny commity pouze pro vybraný projekt. Vybraný projekt je vypsán v textovém poli vlevo nahoře. Nejsou zde žádná tlačítka k ovládání. K přechodu do jiného časového stavu stačí pouze označit požadovaný řádek a dojde automaticky ke změně. Může se stát, že se soubor k vizualizaci dat v nějaké předchozí verzi nevyskytuje, pak dojde k oznámení takovéhoto stavu.

V druhé půlce okna se nachází výběr zdrojů pro vizualizaci. Zdroje jsou zpravidla tři. Prvním je výstup aktuální verze projektu. Ten je defaultně označený, tzn. že se do vizualizace zahrne. Nachází se zde pole, ve kterém je možné upravit popisek grafu. Vše v dolní části zakončuje rozbalovací roletka, ve které můžeme vybrat, jaká data se budou zobrazovat na ose Y. Uprostřed se nachází možnost výběru a přidání reálných dat do vizualizace. Pokud ale nejsou přidané v projektu, vybrat je nelze. Vpravo se nachází druhá část modelu. K výběru verze slouží Git. Na počátku obsahuje stejnou verzi jako v prvním, ale může se to změnit. Funguje stejné vybírání dat pro vizualizaci na osu Y. Stejně tak možnost přepsání popisku do grafu.

Podle zaškrtnutých políček a vybraných názvů dat se, v případě stisku tlačítka *Vizualizace,* dostaneme do pomocných vykreslovacích aplikací, které se otevřou jako nové okno. Pokud se okno zavře, je možné znovu vybírat a dále pracovat s aplikací.

Při přechodu zpět na záložku správy projektů se aplikace vrátí k nejnovější podobě, aby byl zaručen jistý přechod. Toto se případně dá využít jako takový reset k nejnovější verzi.

3.3.2 Programátorská

Seznámíme se s použitými principy a s algoritmy při tvorbě této aplikace. Jak jsme se dočetli v kapitole 3.2.3, aplikace nedisponuje složitou strukturou. Tudíž její popsání není nikterak složité. V podstatě byl založen ve Visual Studiu windows projekt na jádře .NET, tím se automaticky vytvořil čistý formulář se základní inicializací, který lze spustit.

3.3.2.1 Formulář

Formulář obsahuje prvky, které se vkládají z panelu, nebo se mohou vytvářet přímo v kódu. Využité prvky:

- *tab_Control*

  Prvním prvkem, který se vloží na formulář, je prvek se záložkami. Při přechodu mezi jednotlivými záložkami jsou pak na základě událostí přiřazeny kontroly souborů a předávání parametrů.

- *group_Box*

  Na každou záložku se vloží tyto prvky, které mají viditelný nadpis a ohraničují nějakou oblast. Prvky, které se dále vkládají, je možné posunovat najednou. Slouží k vizuálnímu oddělení části aplikace.

- *list_Box*

  List, který obsahuje větší počet věcí stejné struktury. Je možné je spárovat s programovací proměnnou typu *List*. Používá se k výpisu projektů, logu z repozitáře.

- `combo_Box`

  Funguje na stejném principu jako *list_Box*, ale výpis prvků se nachází v rozbalovací roletce, což je ideální prvek pro přehledný výběr zdroje dané simulace k vizualizaci.

- `check_Box`

  Jsou zaškrtávací políčka. Na rozdíl od *radio_Button* je možné jich vybrat více najednou, a to je vhodné pro výběr, co se bude vizualizovat.

- `text_Box`

  Editovatelné pole, do kterého je možné zapisovat znaky, nebo z něj dále číst vepsané informace, zde slouží jako měnitelný popisek grafu ve vizualizačním programu.

- `button`

  Klasické klikací tlačítko, které dostane svůj název a svoji funkci. Tlačítek je v aplikaci využito mnoho.

- `label`

  Jednoduchý textový prvek, který zobrazuje zadanou sérii znaků, nebo celý datový typ *string*, což je řetězec znaků, ukazuje ve vizualizační části, jaký projekt byl vybrán.

3.3.2.2 Globální proměnné

Základní, a všude používané proměnné, jsou založené jako globální, aby je bylo možné využívat v celém projektu. Především se jedná o proměnné typu *BindingList*, které jsou přiřazeny jako zdroj pro *list_Boxy* na formuláři. Tyto proměnné jsou důležité a velmi často využívané.

- `project_List`
- `commit_List`
- `file_commit_List`

Dále proměnné ke vstupnímu souboru, cesta, název, obsah souboru před editací a po ní. Poté cesta do kořenové složky a uložení cest pomocných souborů ke vstupnímu souboru.

3.3.2.3 Metody

Jsou základním stavebním prvkem pro rozsáhlejší kód. Jsou to vlastně funkce v programování, které mohou a nemusí mít návratovou hodnotu. Dále mohou obsahovat vstupní parametry. Z jedné metody se může vyvolat další metoda a může zavolat i sama sebe. Pro zjednodušení bude datový typ *string* zkrácen pouze na *s*.

- `file_Load()`

  Načtení .DAT souboru s uložením názvu, cesty, atd. do proměnných. Dále se vyhledávají cesty k rozšiřujícím souborům.

- `find_Include_Names_In_DAT_File(s cesta)`

  Projde vstupní soubor předaný v parametru *cesta*. Pomocí regulárního výrazu identifikuje cesty a názvy a vloží je do seznamu typu List.

- `copy_Assets()`

  Zkontroluje cesty, případně je vytvoří. Zavolá se metoda *createModifiedDatFile*. Zkopíruje podpůrné soubory pro vstupní soubor na určené místo.

- `createModifiedDatFile(s cesta, s nová_cesta)`

  Začne se číst originální soubor, který se rovnou přepisuje do nového na novém umístění na disku. Ke změně dochází pouze, pokud je obsažena jiná, nežli požadovaná cesta k souborům patřícím ke vstupnímu souboru .DAT.

- `run_BOM()`

  Pokud je nalezen simulátor, pak se spustí samotný proces simulace. Vytvoří se nový commit se spuštěním. Dále proběhne kontrola výstupního souboru, zda neobsahuje chybu, případně zavolá chybovou hlášku.

- *process_Git_cmds(s cestaGit, s text, s cestaDAT)*

  Zkontroluje velikosti potenciálně velkých souborů, které přidá do souboru *.gitignore.* Poté se pomocí metody *RunGitProcess* přidají zbylé soubory do repozitáře a vytvoří se commit. Tím je potřeba znovunačtení seznamu commitů.

- *runGitProcess(s příkaz, s složka)*

  Spustí se proces, ve kterém se provede zadaný git *příkaz*. Složka je cestou na disk, kde se nachází repozitář, a v ní se proces spustí.

- *loadExistingAssets()*

  Vymaže list pro projekty a pomocí vyhledávání ve složce *assets* je znovu přidá.

- *getLastProjectIndex()*

  Podle číslic u složek s projekty vrátí nejvyšší číslici. Využívá se k tomu regulárního výrazu.

- *returnPreviousListBoxIndex(int index, ListBox)*

  Vrací předchozí index z listboxu. Pokud není k dosažení, vrátí nulu.

- *launchSimDialog()*

  V případě, že dojde k úpravě ve vstupním souboru, je zavoláno nové okno, které se zeptá, zda se spustí simulace.

- *loadExistingCommits()*

  Jako u assets se vymaže list, načte se log, který se musí dále předpřipravit.

- *parseGitLog(s log, list commity)*

  Čtení logu a rozdělení jednotlivých údajů do instance třídy *Git_Commit*. Když se údaje naplní, přidá se položka do listu s commity.

- `startsWithHeader(s řádek)`

  Rozlišuje důležitost řádku, dle jeho počátečních znaků, pro *parseGitLog*.

- `parseTBLFile(s soubor)`

  Ve výstupním souboru s příponou *.TBL* se zjišťují hlavičky tabulky pro jednotlivé sloupce, které uloží do Listu.

- `writeToGRFIle()`

  Zapisuje řádky do souboru s příponou ._gr, aby došlo ke správné kombinaci vstupů do vizualizačních programů.

- `startUpCheck()`

  Kontroluje veškeré důležité programy a soubory, potřebné k plné funkčnosti aplikace.

3.3.2.4 Události

Prvky tvoří události, které mají určitý obslužný kód, jenž se vykoná. Většinou obsahuje volání metod nebo jinou jednoduchou obsluhu. Událost může být vyvolána stiskem tlačítka u myši na nějakém prvku nebo zavření okna, které vyskočilo. V ukázce se podíváme na tlačítka, zbytek událostí je velmi obdobný. Seznam kliknutí na jednotlivá tlačítka:

- `btn_Load_Click`

  Pokud se povede načíst soubor, pustí se *copy_Assets*. Načtou se veškeré assets a označí se poslední v listu.

- `btn_Edit_Click`

  Načte se index mezi projekty. Uloží se celý vstupní soubor do proměnné. Spustí se proces s poznámkovým blokem, ve kterém je načtený vstupní soubor. Načte assets a při zavření vrátí index zpět.

- `btn_Add_Click`

  Otevře se dialogové okno pro výběr textových souborů. Pokud se soubor vybere, pak se přidá do konkrétního projektu. Nakonec se opět načtou assety.

- `btn_LaunchBOM_Click`

  Spustí se metoda *run_BOM*. a přidá se commit.

- `btn_Refresh_Click`

  Uloží si aktuální vybraný index v projektech. Spustí se metoda *loadExistingAssets* a poté *loadExistingGitCommits*. Nakonec se vrátí označení tam, kde bylo.

- `btn_GetBack_Click`

  Pokud existuje v repozitáři commit, pak se spustí *RunGitCommits*. Nakonec se obnoví seznamy projektů a git commitů.

- `btn_GetLatestCommit_Click`

  Obdobně jako tlačítko *btn_GetBack*. Změní se pouze parametry metody.

- `btn_Visual_Click`

  Vytvoří se soubor s příponou ._gr dle vybraných checkboxů. Na základě toho se spustí proces *dgraf.cmd*., a tím se spustí vizualizační program.

4 Závěr

Na počátku bylo nejdůležitější pochopit aktuální práci lidí se simulátorem BOModel, jak funguje vstupní soubor a jeho podpůrné soubory, jak se simulátor chová v případě nevalidního vstupu. Kde se dá získat informace o chybách. Kam se ukládají výstupní soubory s výsledky simulace. Po vyzkoušení si různých variant, jsem musel vymyslet strukturu adresářů. Kde se bude jaký program nacházet, kam se budou ukládat jednotlivé projekty. Co bude v sobě každý projekt obsahovat.

Z požadavku ukládání si starších podob jednotlivých projektů, a celkově s vracením se v čase, se nabídlo použití nejrozšířenějšího verzovacího systému Git. Tato platforma je vhodná jak pro použití v samotném uživatelském rozhraní, tak i pro samotný vývoj celé této aplikace.

Jádrem bylo naprogramování aplikace, která bude usnadňovat práci se simulátorem, s využitím znalostí programovacího prostředí Visual Studio a programovacího jazyka C# s pomocí rozšiřujících knihoven včetně základních ovládacích prvků. S tím je svázaná i grafická stránka a její detailní rozvržení. V této oblasti je mnoho možností ke konfiguraci a ve výsledku je možné, že dojde v budoucnu ke změnám. Nicméně v tento moment je vše plně funkční a připravené k provozu dle zadaných požadavků.

Ve vývoji této aplikace se pravděpodobně bude pokračovat dle požadavků koncových uživatelů. Tuto aplikaci je tedy možné dále optimalizovat a rozšiřovat k rozsáhlejšímu využití.

Literatura

[1] ZÁKOPČAN, Marián. *Podzemní Zásobníky Plynu: Postgraduální a inovační studium „Průzkum, těžba a uskladňování kapalin a plynů“*. Hodonín, 2003.

[2] DAKE, LP. *Fundamentals of reservoir engineering* [online]. Seventeenth impression. ELSEVIER SCIENCE B.V., 1998 [cit. 2020-05-27]. ISBN 0-444-41830-X. Dostupné z: https://www.academia.edu/28070833/FUNDAMENTALS_OF_RESERVOIR_ENGINEERING_LP_Dake_.pdf

[3] *Git* [online]. Copyright © [cit. 02.04.2020]. Dostupné z: https://git-scm.com/images/logos/downloads/Git-Logo-2Color.png

[4] CHACON, Scott. *Pro Git* [online]. Praha: CZ.NIC, 2009 [cit. 2020-05-27]. CZ.NIC. ISBN 978-80-904248-1-4. Dostupné z: https://knihy.nic.cz/files/edice/pro_git.pdf

[5] *Git* [online]. Dostupné z: https://git-scm.com/book/en/v2/images/areas.png

[6] [online]. Copyright ©ht [cit. 05.04.2020]. Dostupné z: https://miro.medium.com/max/2000/1*tnvRls6Dg7vFt0zGdtfu_w.png

[7] *302 Found* [online]. Dostupné z: https://i.stack.imgur.com/nWYnQ.png

[8] Git vs. SVN: Which version control system is right for you? - Backlog. *Online Project Management Software for Developers | Backlog* [online]. Copyright © 2020 Nulab, Inc. All rights reserved. [cit. 22.05.2020]. Dostupné z: https://backlog.com/blog/git-vs-svn-version-control-system/

A Obsah přiloženého CD

Přiložené CD obsahuje:

- *text bakalářské práce*

  BP_Victor_Trnka_2020.docx

  BP_Victor_Trnka_2020_STAG.pdf

- *aplikace*

  Spustitelný soubor BOModelUI.exe

  Návod README.md

  Doplňující soubory

- *projekt se zdrojovými kódy*