EduVirtual (Tím číslo 4)

Metodiky projektu

Roly členov tímu, zodpovednosti:

Koník Kristián

- *Manažérske úlohy:*
  - Kontrola stavu systému na správu verzií (Github)
  - Technická dokumentácia projektu
  - Tvorba celkového pohľadu na systém
  - Komunikácia spojená s technickými aspektami projektu
- *Technická práca na projekte:*
  - Vývoj aplikácie virtuálnej reality
  - Návrh a vývoj mobilnej aplikácie s rozšírenou realitou
  - Návrh hrateľnosti

Nagy Adrián

- *Manažérske úlohy:*
  - Návrh a tvorba príručiek
  - Správa a archivácia informácií zo stretnutí
- *Technická práca na projekte:*
  - Vývoj webovej časti aplikácie virtuálnej reality
  - Tvorba množín 3D modelov
  - Podpora grafických vlastností projektu

Pastierovič Dominik

- *Manažérske úlohy:*
  - Code review
  - Tvorba zápisníc zo stretnutí a ich kontrola
  - Tvorba dokumentácie k inžinierskemu dielu (hlavne k implementácii, testovaniu a odstraňovaniu chýb)
  - Návrh metodík programovania
- *Technická práca na projekte:*
  - Vývoj aplikácie virtuálnej reality
  - Návrh testov
  - Testovanie a identifikácia nedostatkov
  - Oprava chýb

Paulen Valentín

- *Manažérske úlohy:*
  - Teamleader, Scrum master

    - Rozdelenie rolí a zodpovedností, aplikácie manažmentov
    - Správa metodík, dohľad nad ich dodržiavaním
    - Dokumentácia manažérskych častí, sprint reviews
    - Podpora komunikácie v tíme, komunikácia s osobami mimo tímu
    - Určenie globálnych cieľov projektu, sledovanie ich napĺňania
- *Technická práca na projekte:*
    - Správa vzdialeného servera cloudtp.fiit.stuba.sk, databázového softvéru
    - Správa externých systémov spolupracujúcich s vytváraným projektom, tvorba modulov na komunikáciu s nimi
    - Návrh funkcionality, zlaďovanie rôznych častí projektu

Pavlenko Dan

- *Manažérske úlohy:*
    - Tvorba dokumentácie k inžinierskemu dielu
    - Dokumentácia k analýze a návrhu modularity projektu
    - Dohliadanie na korektnosť delenia príbehov na úlohy, odhadovania času a aktuálneho stavu rozpracovaných úloh
- *Technická práca na projekte:*
    - Grafická stránka projektu, dizajn vytvorených aplikácií
    - Používateľské rozhranie, návrh a testovanie použiteľnosti, používateľského zážitku
    - Tvorba modelov, animácií, textúr
    - Vývoj aplikácie virtuálnej reality

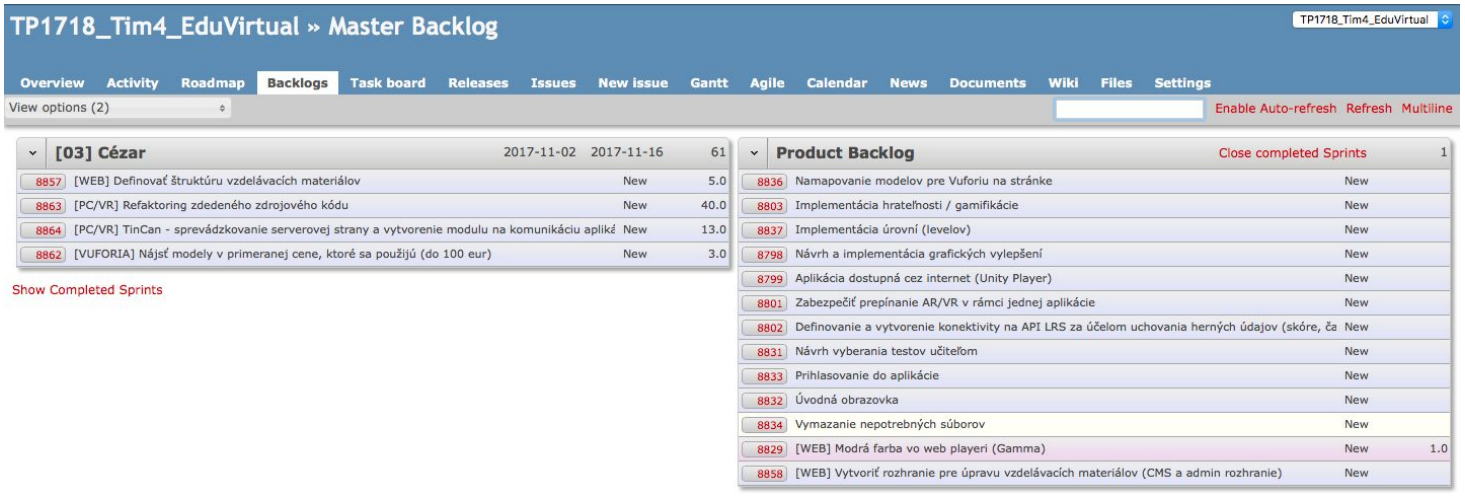
Špuro Ján Jakub

- *Manažérske úlohy:*
    - Vývoj a správa webového sídla projektu
    - Sprístupňovanie informácií o projekte
    - Sumarizácia šprintov a globálna retrospektíva
- *Technická práca na projekte:*
    - Návrh štruktúry webovej časti projektu
    - Vývoj webovej časti projektu
    - Návrh a implementácia databázových modelov
    - Testovanie použitia vytvorených databáz

Metodiky práce so systémom manažmentu projektu Redmine:

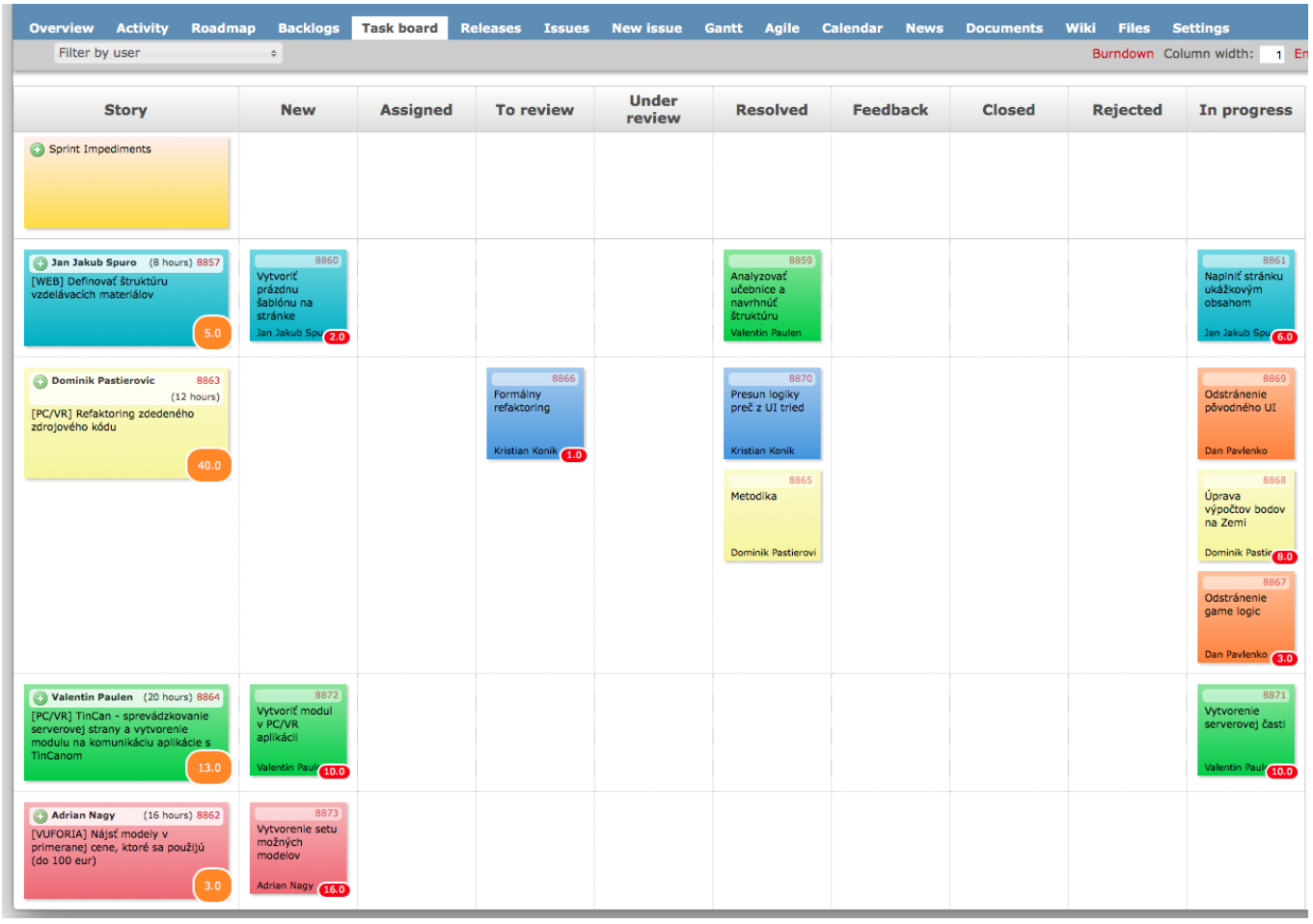
- Systém sa používa na manažment scrumu
- Hlavnou časťou je Product Backlog, v ktorom sú zapísané príbehy (stories)
- Tieto stories je možné deliť do vhodných epicov - väčších skupín združujúcich stories s podobným zámerom

- Každý príbeh (story) by mal mať merateľný prínos pre projekt, mal by byť dobre dokumentovateľný a mal by sa zhodovať s niektorou požiadavkou alebo očakávaním product ownera
- Product Backlog sa tvorí a upravuje na stretnutí s product ownerom
    - So systémom Redmine pracuje na stretnutí jeden člen tímu (podľa kolektívnej dohody, preferovane Scrum master), pričom jeho práca by mala byť viditeľná všetkým
    - Tím na základe konzultácie s vedúcim a product ownerom vytvorí príbehy, prípadne ich zaradí do zodpovedajúcich epicov
    - Následne sa ohodnotí zložitosť každého príbehu
        - Používa sa jednotka Story point
        - Zložitosť sa ohodnocuje technikou “pokeru” - každý člen si vyberie kartu, ktorá zodpovedá jeho odhadu zložitosti
        - Všetci členovia naraz ukážu svoje karty
        - Následne sa diskutuje, prečo členovia zvolili práve také odhady (hlavne minimá a maximá)
        - Po ukončení diskusie sa opakuje vyberanie karty až dovtedy, kým sa všetci nezhodnú na jednej hodnote story pointov - vtedy sa táto hodnota pridelí príbehu a pokračuje sa na ďalší príbeh
    - Ak je niektorý príbeh ohodnotený na 21 a viac story pointov, je vhodné zvážiť rozdelenie na menšie a jasnejšie príbehy
    - Product owner usporiada príbehy podľa priority od najdôležitejšej po najmenej dôležitú
        - Ak je niektorý príbeh ohodnotený moc vysokým počtom story pointov a nevošiel by sa do najbližšieho šprintu, product owner môže navrhnúť rozdelenie príbehu na menšie časti



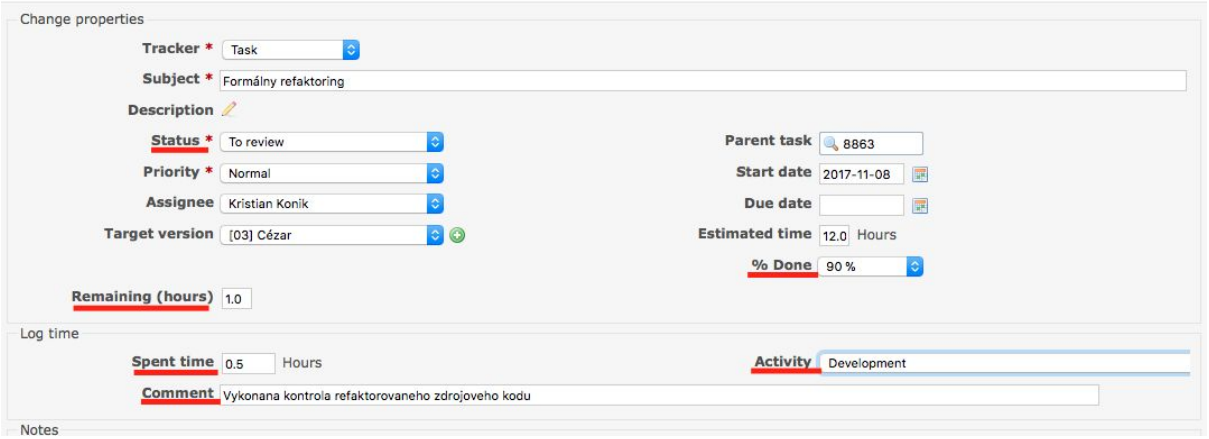
Obrázok 1: Zobrazenie príbehov v product backlogu a v šprinte

- Keď je Product Backlog usporiadaný korektne, vytvorí sa v systéme nový šprint a do neho sa presúvajú príbehy z product backlogu (obrázok 1) podľa priority tak, aby sa celkové množstvo story pointov čo najviac priblížilo určenej hranici
    - Hranicu si určuje tím pri každom vytváraní šprintu podľa výsledkov minulého šprintu, šprint reviewu, prípadne celkového časového plánu na najbližšie 2 týždne
        - Vhodná je snaha postupne zvyšovať hranicu
- Následne sa príbehy pridelia členom, ktorí budú za ne zodpovední, definuje sa DOD
- Príbehy sa rozdelia na úlohy, tiež sa pridelia členom tímu, definuje sa DOD a všetky podrobnosti potrebné pre vypracovanie úlohy (obrázok 2)
- Určia sa časové odhady pre každú úlohu a člen zodpovedný za vykonanie review úlohy (ak je to potrebné)
- Novovytvorené úlohy sa označia stavom New



Obrázok 2: Rozdelenie príbehov na úlohy, pridelenie úloh členom a časový odhad

- Keď člen tímu začne riešiť svoju úlohu, zmení jej stav na In progress
- Po dosiahnutí pokroku v danej úlohe si riešiteľ zaznamená čas (počet hodín, ktoré na úlohe odpracoval), typ práce (Development, Design alebo Documentation) a komentár (obrázok 3)
    - Komentár obsahuje zhrnutie vykonanej práce - stručné, ale úplné, priamočiare a pochopiteľné
- Následne riešiteľ upraví časový odhad zostávajúcej práce na úlohe a počet percent vyjadrujúci dokončenie úlohy



Obrázok 3: Zaznamenanie pokroku v úlohe

- Po dokončení úlohy zmení riešiteľ jej stav na To review
    - Osoba zodpovedná za review zhodnotí výsledok úlohy, vyznačí nedostatky a informuje o nich riešiteľa
    - Riešiteľ nastaví stav úlohy na In progress, nedostatky vyrieši a zmení stav opäť na To review
    - Tento cyklus sa opakuje, pokiaľ review neprebehne bez nedostatkov a DOD bude považované za splnené, vtedy riešiteľ úlohe nastaví stav Resolved
- Pri uzatváraní šprintu tím postupne prezentuje product ownerovi výsledky
- Ak vedúci schváli dokončenie úlohy a splnenie DOD, zmení sa stav úlohy na Closed
- Toto sa opakuje pre všetky úlohy v danom šprinte
- Príbehy, ktoré majú všetky úlohy uzavreté sa považujú za dokončené
    - Tie, ktoré neboli dokončené sa vrátia do Product Backlogu
- Ak sú všetky príbehy v Sprint backlogu dokončené (a nedokončené sú presunuté do produckt backlogu), šprint sa uzavrie
- Pokračuje sa plánovaním ďalšieho šprintu (viď. začiatok tejto kapitoly)

Metodiky práce s GitHubom:

Pri všetkých repozitároch sa dbá na používanie zhodnej formy commit správy, prípadne správy v pull requeste. Táto správa obsahuje (v uvedenom poradí):

- K akému problému / úlohe sa commit viaže (meno, ak nie je, tak stručný a zreteľný popis)
- Čo mal commit riešiť
- Stav, v akom je commitovaná verzia zdrojového kódu
    - Čo bolo vyriešené
    - Čo zostalo nevyriešené
- Commit správy sa preferovane píšu v angličtine

Repozitár s hrou (eduvirtual-globe):

- Vetva **master** - obsahuje odprezentované a schválené verzie aplikácie
    - Commit je možný iba po konci sprintu, ak product owner potvrdí splnenie definition of done
    - Vo výnimočných prípadoch je možný commit aj inokedy, ak sa zistí chyba v aplikácii, ktorú je treba rýchlo odstrániť
    - Každý commit, ktorý splnil DOD, je označený tagom vX.Y, kde Y sa zvýši po splnení DOD a potvrdení od product ownera, X sa zvyšuje pri vykonaní zásadnej zmeny / pridaní dôležitej funkcionality
- Vetva **devel** - obsahuje rozpracovanú verziu aplikácie, do ktorej je možné vykonávať commity po vyriešení úlohy / po dokončení ucelenej funkcionality
- V prípade práce na väčšej zmene / vývoji zložitejšej funkcionality si zodpovedný vývojár vytvorí vetvu z devel vetvy, označí ju názvom, z ktorého bude jasne vyplývať účel takejto vetvy (preferovaný je underscore-case)
    - Po ukončení práce na funkcionalite sa vyhradená vetva spojí (merge) s devel vetvou
    - Spája ju buď vývojár, alebo teamleader podľa dohody
    - So spojením musia súhlasiť všetci zainteresovaní členovia
    - Po úspešnom spojení sa vyhradená vetva odstráni

Repozitár s webovou stránkou edukačnej stránky (eduvirtual-eduweb):

- Vetva **master** - obsahuje dokončené verzie stránky
    - Commit je možný vykonať v prípade potreby po pridaní ucelenej funkcionality
- Počas vývoja zložitejšej časti stránky sa vytvorí vyhradená vetva z vetvy master

    - Po ukončení práce na funkcionalite sa vyhradená vetva spojí (merge) s master vetvou
    - Spája ju buď vývojár, alebo teamleader podľa dohody
    - So spojením musia súhlasiť všetci zainteresovaní členovia
    - Po úspešnom spojení sa vyhradená vetva odstráni

Repozitár s webovou stránkou projektu (eduvirtual-web):

- Vetva **master** - obsahuje dokončené verzie stránky
    - Commit je možný vykonať v prípade potreby po pridaní ucelenej funkcionality
- Počas vývoja zložitejšej časti stránky sa vytvorí vyhradená vetva z vetvy master
    - Po ukončení práce na funkcionalite sa vyhradená vetva spojí (merge) s master vetvou
    - Spája ju buď vývojár, alebo teamleader podľa dohody
    - So spojením musia súhlasiť všetci zainteresovaní členovia
    - Po úspešnom spojení sa vyhradená vetva odstráni

Ďalšie repozitáre:

- V prípade potreby je možné operatívne vytvárať ďalšie repozitáre
- Pomenovanie repozitárov: eduvirtual-*TYPE_SPECIFICATION*
    - TYPE značí obsah, ktorý sa v repozitáry nachádza (globe pre VR hru, web pre stránku projektu, eduweb pre edukačnú stránku a pod.)
    - SPECIFICATION spresňuje, k čomu repozitár slúži (legacy pre uschovanie zdedeného kódu, revXY pre uloženie revízie a pod.)
- Spôsob tvorby a označovania commitov je zhodný s ostatnými repozitármi, práca s vetvami sa určí dohodou a umiestni sa do wiki stránky repozitára
- Každý člen, ktorému je pridelené prístupové právo je oboznámený s účelom repozitára a spôsobom práce s repozitárom
- Je možné vytvoriť aj read-only repozitáre (nie je umožnený commit ani tvorba novej vetvy)
    - To je užitočné napríklad pri zdieľaní staršej verzie zdedeného kódu - tá má mať iba informatívny charakter

Metodiky komunikácie:

Tímový mail:

- fiit_tp1718_team04@outlook.com
- Tento mail slúži na komunikáciu s product ownerom a externými osobami
- Prístup k účtu má každý člen tímu
- Nové maily píše hlavne teamleader po dohode s ostatnými členmi tímu

    - V prípade potreby môže napísať nový mail ktokoľvek s tímu, pričom o tom informuje ostatných členov tímu čo najskôr
- Predmet novej správy má formu [TP] Téma - kde Téma je výstižný názov obsahu mailu
- Podpis na konci má formu Meno Priezvisko, tím 4, kde Meno a Priezvisko je meno a priezvisko autora mailu
- Outlook automaticky zakončuje maily pätičkou:

  Tim c. 4
  EduVirtual
  Timovy projekt FIIT STU

Slack:

- tim4eduvirtual.slack.com
- Slack sa používa na komunikáciu a zdieľanie dát medzi členmi tímu
- Správy adresované všetkým členom sa vkladajú do kanálu #general
- Pre komunikáciu s iba jedným členom sa používajú priame správy

Zdieľanie súborov cez Google Drive:

- Pre zdieľanie a zálohovanie dokumentov a súborov, ktoré sa nezaraďujú do žiadneho Github repozitára, je vytvorený zdieľaný priečinok TP_share na Google Drive účte teamleadera
- Plný prístup k nemu má každý člen tímu
- Vkladať súbory môže každý člen tímu podľa potreby
    - Jednotlivé súbory je možné vkladať priamo do koreňového adresára, viaceré súvisiace súbory sa vkladajú do nového priečinka
    - Názvy súborov alebo priečinkov musia výstižne popisovať obsah
- Po vložení súboru by mal autor informovať všetkých ostatných členov tímu

Metodiky pre písanie zdrojového kódu:

C# development

- Najdôležitejšie pravidlo: pokiaľ nerobím grafické výpočty ktoré sa spúšťajú každý frame tak čitateľnosť kódu a jeho pochopiteľnosť sú dôležitejšie ako dobrý výkon. Ak robím kód náročný na výkon, môžem ho napísať horšie pochopiteľne ale treba okomentovať čo a ako kód robí!!!
    - Nesprávne:

```
public static double GetDistance(GpsCoordinates point1, GpsCoordinates point2)
{
    return GlobalConstants.EARTH_RADIUS_METERS * 2 * Math.Atan2(Math.Sqrt(Math.Sin(ConvertToRadians(point2.Latitude - point1.Latitude) / 2)
        * Math.Sin(ConvertToRadians(point2.Latitude - point1.Latitude) / 2) +
        Math.Cos(ConvertToRadians(point1.Latitude)) * Math.Cos(ConvertToRadians(point2.Latitude)) *
        Math.Sin(ConvertToRadians(point2.Longitude - point1.Longitude) / 2) * Math.Sin(ConvertToRadians(point2.Longitude - point1.Longitude) / 2)),
        Math.Sqrt(1 - Math.Sin(ConvertToRadians(point2.Latitude - point1.Latitude) / 2) * Math.Sin(ConvertToRadians(point2.Latitude - point1.Latitude) / 2) +
        Math.Cos(ConvertToRadians(point1.Latitude)) * Math.Cos(ConvertToRadians(point2.Latitude)) *
        Math.Sin(ConvertToRadians(point2.Longitude - point1.Longitude) / 2) * Math.Sin(ConvertToRadians(point2.Longitude - point1.Longitude) / 2)));
}
```

    - Správne
-
-
- Napísať summary funkcie (Visual studio po napísaní /// automaticky doplní snipet s popisom funkcie, existujúcimi argumentmi a návratovou hodnotou. Toto summary sa následne zobrazuje pri mouseoverovaní

```
public static double GetDistance(GpsCoordinates point1, GpsCoordinates point2)
{
    float radius = GlobalConstants.EARTH_RADIUS_METERS;

    double latitude1Rad = ConvertToRadians(point1.Latitude);
    double latitude2Rad = ConvertToRadians(point2.Latitude);

    double deltaLatitudeRad = ConvertToRadians(point2.Latitude - point1.Latitude);
    double deltaLongitudeRad = ConvertToRadians(point2.Longitude - point1.Longitude);

    double a = Math.Sin(deltaLatitudeRad / 2) * Math.Sin(deltaLatitudeRad / 2) +
            Math.Cos(latitude1Rad) * Math.Cos(latitude2Rad) *
            Math.Sin(deltaLongitudeRad / 2) * Math.Sin(deltaLongitudeRad / 2);
    double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));

    return radius * c;
}
```

  funkcie/parametrov, vďaka čomu je možné pochopiť čo funkcia vyžaduje a čo spraví bez študovania jej kódu.

```
/// <summary>
/// Calculates flight distance (earth-curved trajectory ignoring hills) between two longtitude and latitude defined points on Earth
/// </summary>
/// <param name="point1">Starting point</param>
/// <param name="point2">Point to where we want to calculate distance from starting point</param>
/// <returns>Distance from one point to another in kilometers</returns>
public static double GetDistance(GpsCoordinates point1, GpsCoordinates point2)
```

- POUŽÍVAŤ { get; set; } namiesto java getter setter. get a set môžu obsahovať aj implementáciu ale väčšinou ju netreba. Ak ju treba (príklad):
    - Verejný getter, privátny setter, getter obsahuje úpravu údaju
    - double debt { public get { return Math.Round(field, 2); }; private set; }
    - Ak je logika príliž rozsiahla, je možné ju napísať ako funkciu na viac riadkov.
- Názvy:
    - Triedy: čo najkratšie názvy (public class UpperCamelCase{})
    - Privátne premenné, lokálne premené: začínajú s malým písmenom, ostatné začiatky slov s veľkým (private int lowerCamelCase;)
    - Funkcie: UpperCamelCase
    - Interface: I<UpperCamelCase>
- Neinicializovať fieldy v classach, sú automaticky nastavené na 0/null/false. Pokiaľ potrebujeme inú hodnotu, ktorá nezávisí od argumentov konštruktorov, vtedy môžeme použiť inicializáciu pri deklarácii.
- Operátory: oddeľovať medzerou (1 + x) * 3 || y << 2 namiesto (1+2)*3||y<<2
- Blokové zátvorky na samostatný riadok

- Pokiaľ mám príliš dlhý riadok, rozdeliť ho na viac s odsadením jedného tabu:

```
double a = Math.Sin(deltaLatitudeRad / 2) * Math.Sin(deltaLatitudeRad / 2) +
        Math.Cos(latitude1Rad) * Math.Cos(latitude2Rad) *
        Math.Sin(deltaLongitudeRad / 2) * Math.Sin(deltaLongitudeRad / 2);
```

- Pri konvertovaní primitívnych typov (int->double, float->string) použiť namiesto double x = (double)y; class Convert -> var x = Convert.ToDouble(y). Convert je lahšie debugovateľný pri prípadnej strate dát, ľahšie sa chápe zámer kódu.
- Používať var namiesto explicitného názvu premennej pokiaľ pravá strana výrazu jednoznačne určuje typ výsledku:
    - foreach (var leaf in Tree.Leafs) namiesto foreach (Leaf leaf in Tree.Leafs)
- Garbage collector odstráni objekt len ak na neho neexistuje referencia. Ak pridáte objekt do listu, nezabudnite ho odstrániť.
- C# má JIT compiler (Just in Time), čo znamená, že kompiluje kód za behu programu. Toto umožňuje optimalizáciu strojového kódu podľa hardvérovej špecifikácie systému, na ktorom je program vytvorený. Ak vytvárame veľké bloky kódu, strácame výhody JIT compileru a spôsobujeme pomalší beh programu (musí sa dlho čakať na kompiláciu). JIT kompiluje celý kód funkcie. Funkcie by mali byť čo najkratšie. Ak je vo funkcií veľa if logiky, je lepšie ju nahradiť funkciami alebo objektami. Ak sa funkcionalita opakuje, treba ju vytiahnuť do samostatnej funkcie, pretože táto funkcia po skompilovaní už bude pripravená pri každom jej volaní, namiesto kompilácie tejto logiky na rôznych miestach ako podčasť inej funkcie.

HTML / CSS / JS development

Názvy súborov, priečinkov, tried a identifikátorov uvádzame malým písmom v angličtine a medzery nahrádzame znakom “_” prípadne “-”. Snažíme sa však udržiavať rovnakú konvenciu v názvoch súborov.
Príklad: **example_main.css** alebo **example-main.css**

Organizácia súborov:

- Písma v priečinku **fonts**
- CSS štýly v priečinku **css**
- JavaScripty v priečinku **js**
- Obrázky v priečinku **images**
- Videa v priečinku **videos**
- v prípade ak sa dajú vyššie spomínané súbory ďalej organizovať do podpriečinkov tak sa je vhodné ich takto organizovať, v prípade iných súborov postupujeme analogicky

Metodiky pre HTML a CSS:

- Pri písaní HTML kódu usilujeme o dodržiavanie atribútov pre HTML5 a zachovanie sémantiky kódu - elementy, ktoré špecifikujú svoj obsah ako je <form>, <table>, <article>, , … by mali obsahovať len obsah na, ktorý sú určené
- Kaskádové štýly (CSS) ukladať do externých súborov mimo HTML v separátnom priečinku s názvom **css**, rovnako sa vyhýbať inline štýlom ak je to možné, definovanie pozadia (background-image) a podobne sa môžu vyskytovať
  Príklad **inline** štýlu: *** Text ***
- Názvy štýlov by mali byť organizované do hierarchie ako **.parent .child .subchild** { /* Some stuff… */ } namiesto .**parent-child-subchild** { /* Some stuff… */ }, tento spôsob umožňuje jednoduchšie vykonanie prípadných zmien a šetrí miesto v HTML kóde. Príklad je možné vidieť na nasledujúcom obrázku:

NESPRÁVNE

```html
<html>
  <style>
    /* Some CSS */
    .parent {
      display: block;
      width: 80%;
      height: auto;
    }
    .parent-child {
      display: inline-block;
      width: auto;
      height: auto;
    }
    .parent-child-subchild {
      display: inline-block;
      width: auto;
      height: auto;
    }
  </style>
  <body>
    <div class="parent">
      <div class="parent-child">
        <div class="parent-child-subchild">Lorem ipsum...</div>
      </div>
    </div>
  </body>
</html>
```

SPRÁVNE

```css
/* exmaple.css */
.parent {
  display: block;
  width: 80%;
  height: auto;
}
.parent .child {
  display: inline-block;
  width: auto;
  height: auto;
}
.parent .child .subchild {
  display: inline-block;
  width: auto;
  height: auto;
}
```

```html
<html>
  <link rel="stylesheet" type="text/css" href="css/example.css">
  <body>
    <div class="parent">
      <div class="child">
        <div class="subchild">Lorem ipsum...</div>
      </div>
    </div>
  </body>
</html>
```

- Podľa možnosti využívame CSS atribúty, ktoré sú kompatibilné s viacerými prehliadačmi a vhodné je taktiež definovanie fallback atribútov - napríklad pre písma “font: 12px Roboto-Light, Arial” ak nie je možné použiť písmo **Roboto** bude namiesto neho použité písmo **Arial**
- Veľké množstvo atribútov je možné zapísať v skrátenom tvare do jedného riadku, ak je to možné a nespôsobuje to neprehľadnosť využívame skrátený

tvar. Príklad:

DOBRÉ

```css
.parent {
  display: block;
  width: 80%;
  height: auto;

  border-width: 1px;
  border-style: solid;
  border-color: #f9e9d3;
}
```

LEPŠIE

```css
.parent {
  display: block;
  width: 80%;
  height: auto;

  border: 1px solid #f9e9d3;
}
```

- Vrámci CSS štýlu oddeľujeme súvislé bloky voľným riadkom pre lepšiu prehľadnosť
- Pri mapovaní štýlu preferujeme triedy
  CSS "**.parent**" -> HTML "<p **class="parent"**></p>" pred identifikátormi
  CSS "**#parent**" -> HTML "<p **id="parent"**></p>"
- Ak je to možné, minimalizujeme počet definovaných tried tak, že uvedieme názov tagu namiesto definovania triedy tak ako je možné vidieť na nasledujúcom obrázku:

DOBRÉ

```css
/* Some CSS */
.parent {
  display: block;
  width: 80%;
  height: auto;
}
.parent .link {
  color: red;
  text-decoration: none;
}

<html>
  <body>
    <div class="parent">
      <a href="example.html" class="link">Some link</a>
    </div>
  </body>
</html>
```

LEPŠIE

```css
/* Some CSS */
.parent {
  display: block;
  width: 80%;
  height: auto;
}
.parent a {
  color: red;
  text-decoration: none;
}

<html>
  <body>
    <div class="parent">
      <a href="example.html">Some link</a>
    </div>
  </body>
</html>
```

Metodiky pre písanie JavaScript:

- Názvy súborov, metód, objektov, premenných uvádzame v angličtine.
- Okrem inicializácie objektov / funkcií sa musia všetky kódy nachádzať v externých súboroch v separátnom priečinku s názvom **js** - podporuje to jednoduchšie vykonávanie zmien, kde stačí upraviť jeden JS skript namiesto všetkých HTML súborov, kde sa nachádza daný interný skript - podla možnosti by sa však aj inicializačné skripty mali nachádzať v externom súbore
- Pri programovaní skriptov využívame **objekty** kde sa dá namiesto procedurálneho prístupu
- Pri názvoch premenných a funkcií sa využíva **camelCase**
- Pri definovaní názvov "tried" využívame **PascalCase**

- Názvy globálnych premenných a konštánt uvádzame VEĽKÝM_PÍSMOM a ak sa jedná o viac slov oddelíme ich znakom "_"
- Medzi operátormi nechávame medzeru: a = b; a = b + 3; if (a > b)

SQL development

- Entity sa pomenúvajú po anglicky, názvom, ktorý čo najjasnejšie vystihuje ich zmysel alebo funkciu
    - Využíva sa PascalCase - názov sa začína veľkým písmenom, pokračuje malými, nové slovo sa napája bez medzery na to predchádzajúce, ale prvé písmeno nového slova je veľké
    - Príklad: User, Course, CourseLevel, CourseUser
- Atribúty v entitách sa píšu malými písmenami, slová sa oddeľujú podtržníkom
    - Príklad: id, user_id, bounding_box

Metodiky pre písanie dokumentácie:

Dokumentácia zo stretnutí

- Zápisnica sa vytvorí čo najskôr po každom stretnutí tímu (ešte v deň stretnutia)
- V ten istý deň sa zápisnica uverejní na stránke tímu vo formáte PDF
- Počas stretnutia sa určí člen tímu zodpovedný za vytvorenie zápisnice a člen zodpovedný za jej revíziu a umiestnenie na webovej stránke tímu
- Názov súboru je zapisnica_stretnutie_X.pdf, kde X je poradové číslo stretnutia
- Štruktúra dokumentu (zápisnice):
    - Nadpis: Zápisnica stretnutia tímu EduVirtual (tím číslo 4)
    - Téma stretnutia
    - Dátum stretnutia
    - Miesto stretnutia (zvyčajne FEI STU B402)
    - Zoznam prítomných a neprítomných členov (vrátane vedúceho)
    - Meno zapisovateľa zápisnice
    - Program stretnutia
        - V odrážkach napísané témy, okruhy alebo konkrétne otázky, ktoré sa na stretnutí riešili
        - Každý bod musí byť pochopiteľný
            - ak nie je účel témy jasný z názvu, je potrebné ho niekoľkými vetami upresniť
        - Voliteľne je možné uviesť aj meno člena, ktorý navrhol daný bod zaradiť do programu
    - Vyhodnotenie úloh z predchádzajúceho stretnutia - Backlog ukončené šprintu (ak počas stretnutia šprint nekončí, uvedie sa namiesto nižšie

popísanej tabuľky informácia “Počas tohto stretnutia šprint ešte prebiehal, úlohy teda neboli uzavreté. Aktuálny stav úloh a priebežné výsledky sme prezentovali vedúcim a zapísali sme pripomienky.”)

- ide o tabuľku obsahujúcu stories alebo úlohy daného šprintu vo forme:
  - ID - identifikátor story alebo úlohy v nástroji na manažovanie projektu
  - Názov - meno story alebo úlohy
  - Členovia - na prvom mieste je uvedené meno člena, ktorý je za danú story alebo úlohu zodpovedný, za čiarkou môžu voliteľne pokračovať mená členov, ktorí sa na riešení story / úlohy podieľajú
    - Táto bunka tabuľky musí obsahovať minimálne jedno meno: zodpovedného člena
    - V prípade, že na riešení story / úlohy sa podieľajú všetci, členovia, uvedie sa na prvom mieste meno zodpovedného člena a za čiarkou sa uvedie slovo “všetci”
  - Dátum zadania - dátum, kedy bola story / úloha vytvorená
  - Očakávaný dátum ukončenia - dátum, kedy je naplánované dokončenie danej story / úlohy
  - Stav - zhoduje sa so stavom, ktorý je danej úlohe pridelený v nástroji na manažment projektu
    - V zátvorke za názvom stavu sú uvedené percentá vyjadrujúce časť úlohy, ktorá je dokončená
    - V prípade, že je úloha uzavretá a Product Owner potvrdil splnenie Definition of Done, je táto skutočnosť uvedená v zátvorke (DOD splnené)
    - V prípade, že je úloha uzavretá a Product Owner potvrdil jej zrušenie, je v zátvorke uvedené (Zrušená)
    - Za zátvorkou môže byť uvedená krátka poznámka
    - Ak je potrebné upresniť stav, je možné využiť poznámku pod čiarou a do nej umiestniť podrobné vysvetlenie aktuálneho stavu vrátane faktorov, ktoré tento stav zapríčinili

- Opis stretnutia - každý bod z časti Program stretnutia sa podrobne rozpíše
  - Uvádza sa sem, čo zapríčinilo potrebu riešiť tento bod, aký je očakávaný cieľ diskusie, samotný priebeh diskusie so všetkými podrobnosťami a jasne zhrnutý záver
- Aktuálny stav úloh v systéme manažmentu projektu

        - Tu sa uvádza Sprint Backlog práve prebiehajúceho alebo novovytvoreného šprintu
        - Štruktúra tabuľky sa zhoduje s tabuľkou v časti Vyhodnotenie úloh z predchádzajúceho stretnutia, ale bol pridaný jeden stĺpec:
            - Story Points / Odhad. čas - počet story pointov predelený story / úlohe a odhadovaný čas na splnenie úlohy
                - v prípade prebiehajúceho šprintu sa ako odhadovaný čas uvádza, koľko času ešte zostáva
        - V prípade, že sa na stretnutí zmení Product Backlog, uvedie sa v tejto kapitole aj tabuľka s kompletným Product Backlogom, jej štruktúra je nasledovná:
            - ID - identifikátor story alebo úlohy v nástroji na manažovanie projektu
            - Názov - meno story alebo úlohy
            - Zodpovedný člen - člen, ktorému bola story alebo úloha priradená
                - bunka môže byť prázdna (úlohy v product backlogu nemusia byť pridelené)
            - Typ - typ story alebo úlohy uvedený v nástroji na manažovanie projektu

Dokumentácia k retrospektíve šprintu

- Na stretnutiach tímu, počas ktorých sa uzatvára šprint, je potrebné vykonať retrospektívu práve končiaceho šprintu
- Táto retrospektíva má formu otvorenej diskusie medzi všetkými členmi tímu a vedúcimi projektu
    - Téma diskusie je, ako prebiehal šprint, čo sa podarilo a čo sa nepodarilo. Pre každý bod v časti neúspešných vlastností sa kolektívne hľadá konkrétne a realizovateľné riešenie
- Počas retrospektívy vzniká dokument zachytávajúci všetky identifikované body
    - Dokument vzniká priamo na stretnutí tak, že ho všetci vidia a môžu do neho prispieť (počas písania je napríklad zobrazený na projektore)
    - Ide o kolektívne dielo vznikajúce na základe diskusie, píše ho jeden člen, ale jeho meno sa v dokumente neuvádza (keďže dokument je kolektívny)
    - Štruktúra:
        - Identifikácia šprintu (jeho poradové číslo)
        - Dátum, kedy dokument vznikol
        - Nedostatky
            - Ako odrážky
            - Usporiadané podľa úloh

            - K nedostatkom sa priraďujú nájdené riešenia (ako pododrážky)
        - Úspechy
            - Ako odrážky
            - Usporiadané podľa úloh
    - Dokument sa po jeho vytvorení umiestni do tímového Google Drive priečinka (do podzložky sprint_reviews)
        - Prístup na čítanie k nemu majú všetci členovia tímu aj vedúci
        - Zmeny v týchto dokumentoch bežne nie sú povolené, môžu sa vykonávať iba vo výnimočných prípadoch, a to so súhlasom všetkých členov aj vedúceho
        - V prípade, že sa dokument zmení po ukončení stretnutia, na ktorom bol vytváraný, musí byť táto skutočnosť uvedená (spolu s dôvodom zmeny) na konci dokumentu spolu s menom člena, ktorý zmenu vykonal