

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova2, 842 16 Bratislava 4

AugReality

Dokumentácia projektu

Vedúci tímu: Ing. Kapec Peter Phd.

Členovia tímu: Bc. Filip Škultéty,
Bc. Peter Belai,
Bc. Martin Mokrý,
Bc. Lukáš Hagara,
Bc. Patrik Berger,
Bc. Marek Roštár,
Bc. Michal Galbavý

Akademický rok: 2016/2017

Obsah

1. Úvod.....	3
1.1. Podiel práce na dokumentácií k riadeniu projektu.....	3
1.2. Role členov tímu a podiel práce.....	3
2. Aplikácie manažmentov.....	5
3. Sumarizácie šprintov.....	7
3.1. Začiatočná fáza.....	7
3.2. Šprint č. 1.....	7
3.3. Šprint č. 2.....	7
4. Používané metodiky.....	9
5. Globálna retrospektíva ZS.....	9
6. Motivačný dokument.....	9
6.1. Tím.....	9
6.2. Motivácia - Extrakcia dát z webu [WebExtraction].....	10
6.3. Motivácia - Inteligentný sklad [SmartStore].....	10
6.4. Motivácia - Vizualizácia informácií v obohatenej realite [AugReality].....	11
7. Metodiky.....	12
7.1. Ako programovať v C++.....	12
7.2. CppCheck.....	14
7.3. CppLint.....	15
7.4. PlantUML.....	17
7.5. GitFlow.....	22
7.6. Sphinx dokumentácia.....	24
7.7. TFS.....	27
7.8. Logovanie s knižnicou Easylogging++.....	30
7.9. Metodika na písanie BDD testov.....	31
8. Export evidencie úloh.....	33
8.1. Šprint č. 1, týždeň 1.....	33
8.2. Šprint č. 1, týždeň 2.....	34
8.3. Šprint č. 2, týždeň 1.....	34

1. Úvod

Kapitola riadenie projektu opisuje všetky procesy, ktoré boli použité pre efektívnejšiu prácu tímu Aug_RealityKings počas tímového projektu. Obsahuje využívané metodiky, pravidlá, postupy, ktorými sme sa riadili. Z dôvodu, že na projekte pokračujeme po minuloročných tímoch/študentoch niektoré časti riadenia sme prebrali, niektoré upravili/doplnili a niektoré vytvorili nové.

V kapitole 1.1 je zoznam kapitol s uvedením autora, ktorý danú kapitolu vypracoval. Určenie a popis rolí sa nachádza v kapitole 1.2. Každému členovi boli určené viaceré role a zodpovednosti. V kapitole 2 je opis aplikovaných metodík, ktorými sa tím riadil. 3. kapitola sumarizuje doterajšie šprinty a úlohy. V 4. kapitole je stručný opis používaných metodík, ktorými sme sa riadili. V 5. kapitole je globálna retrospektíva zimného semestra, ktorá sumarizuje doterajšie šprinty.

1.1. Podiel práce na dokumentácii k riadeniu projektu

Kapitola	Autor/autori
Úvod	Filip Škultéty
Role členov	Filip Škultéty
Aplikácie manažmentov	Filip Škultéty
Sumarizácia šprintov	Filip Škultéty
Používané metodiky: TFS	Filip Škultéty
Používané metodiky: Astyle	Martin Mokrý
Používané metodiky: Cplint	Martin Mokrý
Používané metodiky: Cppcheck	Martin Mokrý
Používané metodiky: Logovanie	Peter Belai
Používané metodiky: Testovanie	Lukáš Hagara
Používané metodiky: GitFlow	Patrik Berger
Globálna retrospektíva zimného semestra	Celý tím
Export úloh počas šprintov	Filip Škultéty
Tvorba stránky	Martin Mokrý

1.2. Role členov tímu a podiel práce

Členovia tímu nadobúdali rôzne role počas semestra, dynamicky pribúdali povinnosti a zodpovednosti členov.

Marek Roštár

Marek je linux integrátor a tester. Keďže projekt je multiplatformový a Marek je jediný člen tímu, ktorý pracuje s operačným systémom Linux, nikto iný z tímu sa nehodil na túto pozíciu viac než on. Úloha linuxového integrátora zahŕňa: testovanie kompatibility implementovaných častí, podpora ovládačov.

Peter Belai

Peter je študentský vedúci tímu. Počas ZS sa venoval tvorbe dátových sad pre ostatných študentov fakulty, ktorý pracujú/budú pracovať s vizualizáciou dát. Dátové sady získaval pomocou zariadenia kinect 1 a snaží sa riešiť získavanie dát zo zariadenia kinect 2.

Filip Škultéty

Filip je manažér dokumentácie. Stará sa o vytváranie exportov zo softvéru na manažment riadenia tímu a generovanie technickej dokumentácie.

Martin Mokry

Martin je zodpovedný za spravovanie webovej stránky, aktualizovanie obsahu, nahrávania nových dokumentov. Počas šprintov 1-3 sa venoval štruktúre programu a tvorbe analýzy nových častí.

Patrik Berger

Patrik spravuje spoločný git repozitár. Spolu s Martinom Mokrym navrhujú architektúru nových častí systému, konkrétne zobrazenie modelov rúk v scéne. Spravovaním úloh na TFS prispieva k manažmentu úloh.

Lukáš Hagara

Lukáš je manažér vývoja a testovania.

Michal Galbavý

Opustil tím v treťom týždni zimného semestra. Nevykonával žiadnu rolu.

2. Aplikácie manažmentov

V kapitole je bližší opis aplikovania manažmentov.

Manažment komunikácie

Na komunikáciu v tíme používame:

- Stretnutie tímu podľa rozvrhu- v ZS semestri stretnutia prebiehali každý štvrtok od 10:00 do 13:00.
- Slack- Mimo tímových stretnutí komunikujeme pomocou nástroja Slack, v ktorom sme vytvorili samostatné sekcie pre dokumentáciu, manažment, oznamy, vývoj a web stránku.

Tvorba dokumentácie

Dôležitou súčasťou softvéru je dokumentácia, ktorú treba neustále dopĺňať a aktualizovať. Metoda SCRUM, ktorou sa riadime, vyžaduje taktiež tvorbu dokumentácie. Počas projektu sme vytvorili alebo dopĺňali nasledovné dokumenty:

- Motivačný dokument
- Zápisnice zo stretnutí
- Exporty zo softvéru na manažment úloh, konkrétne TFS
- Retrospektívy na konci šprintov
- Dokumentácia k riadeniu
- Dokumentácia k inžinierskemu dielu
- Inštalačná príručka
- Technické dokumentácie (sphinx, doxygen, latex)

Manažment verzii

Pred implementáciou novej funkcionality, si vytvárame novú vetvu produktu. Vetvenie zmien, funkcionality umožňuje vrátenie k predchádzajúcim verziám a opravu chýb. Pre správu verzii používame Git. Pri vývoji kladieme dôraz aby boli zmeny komitované do prislúchajúcej vetvy. Keď je funkcionality hotová, vytvárame pull request, po ktorom členovia tímu testujú a vyjadrujú sa k novej funkcionalite a navrhujú schválenie alebo prerobenie.

Proces plánovania úloh

Riadime sa metodikou SCRUM, v ktorej šprint trvá 2 týždne. Na začiatku, respektíve na konci predchádzajúceho šprintu sa plánujú úlohy, ktoré sa budú v danom šprinte vykonávať. Plánovanie úloh zahŕňa ohodnotenie úloh, určenie podmienok, za ktorých sa úloha vyhodnotí ako dokončená a priradenie priorít úlohám. Po vytvorení a nahodení úloh do TFS, prebieha pridelovanie úloh, keď

si každý člen vyberie úlohu/úlohy, za ktorú bude zodpovedný. Na stretnutí po prvom týždni šprintu sa diskutuje pokrok v danom šprinte a riešia sa vyskytnuté problémy. Na zaznamenávanie stavu úloh používame nástroj Team Foundation Server 2015(skrátene TFS). Bližšie informácie o používaní nástroja TFS sa nachádzajú v metodike TFS.

Proces hodnotenia úloh

Úlohy hodnotíme na začiatku šprintu. Úlohy hodnotíme hodnotami 1, 2, 3, 5, 8, 13, 20, 40, 100 a ?. Ak má úloha hodnotenie viac ako 13, rozdelí sa na viacero úloh, aby hodnotenie bolo menšie 13. Ak tím vyhodnotí úlohu rôznymi hodnotami, prebehne diskusia, kde členovia vyjadria, prečo ohodnotili úlohu daným číslom. Cyklus sa opakuje pokým sa tím nezhodne v hlasovaní. Následne vlastník produktu stanoví priority úlohám.

Manažment kvality

Nás vlastník produktu je nás vedúci tímu, ktorému na stretnutí prezentujeme dosiahnutý progres a výstupy úloh. Vyjadrenia, poznámky a pripomienky zapisuje zapisovač stretnutí. Pre zachovanie kvality kódu používame nástroje CppLint, Cppcheck a na jednotné formátovanie zdrojového kódu používame Astyle. Pred dokončením úlohy, je potrebné opraviť upozornenia a chybové hlášky z vyššie spomenutých programov, aby sa mohla úloha vyhodnotiť ako dokončená.

Správa webovej stránky

Na začiatku bolo potrebné vytvoriť tímovú webovú stránku, ktorá sa každý týždeň aktualizuje v závislosti od vytvorených dokumentov. Počas semestra je web správca zodpovedný za udržiavanie obsahu stránky. Keď člen tímu vytvorí dokument, ktorý je určený na publikovanie, pošle tento dokument správcovi webu.

Proces testovania

Testovanie je v našom projekte dôležitou súčasťou. Po programovaní, refaktorizácii, musí prejsť projekt procesom testovania. Počas programovania projektu sa vytvorili dvojice, v ktorých si členovia navzájom kontrolujú a testujú program. Každá nová funkcionálna musí byť otestovaná na Linux aj Windows. Pred ukončením úlohy je potrebné vykonať aj akceptačné a regresné testovanie.

3. Sumarizácie šprintov

3.1. Začiatková fáza

V začiatkovej fáze sme sa oboznamovali s projektom. Aktuálny stav projektu je výsledkom viacerých tímových, diplomových, bakalárskych prác a preto sme úvodne chvíle venovali zorientovaniu sa v stave projekte. V druhom týždni ZS sme sa venovali inštalácií potrebných programov, ovládačov a vykonali sme prvé kompilácie a spustenie programu. Začali sme používať nástroj na manažment úloh – TFS.

3.2. Šprint č. 1

Na začiatku šprintu č. 1 sme diskutovali s vedúcim tímu o prvých úlohách. Prvý šprint sa niesol v znamení metodík- študovanie, upravovanie a vytváranie nových metodík, ktoré nám budú pomáhať efektívne fungovať ako tím. Medzi hlavné úlohy patrilo vytvorenie metodiky ku kvalite, gitflow, TFS, code review, logovaniu, testovaniu. V prvom šprinte prebiehalo aj testovanie existujúcich funkcionalít systému. Vlastníkovi produktu sme ukázali výsledky úloh, ku ktorým sa vyjadril.

Retrospektíva

Na záver šprintu sme diskutovali, ktoré veci treba zlepšiť. Problémom prvého šprintu bolo, že sme sa neriadili všetkými metodikami, lebo sme nemali zosynchronizované dokumenty s metodikami.

Project: AugReality Server: tfs.fiit.stuba.sk\StudentsProjects Query: Report sprintu List type: Tree			
ID	Work Item Title	Assigned To	State
4540 Task	Metodika ku kvalite	Bc. Martin Mokry	Done
4541 Task	Metodika gitflow	Bc. Patrik Berger	Done
4542 Task	Metodika k TFS	Bc. Filip Skultety	Done
4543 Task	Metodika ku code review	Bc. Patrik Berger	Done
4544 Task	Metodika k logovaniu	Bc. Peter Belai	Done
4545 Task	Metodika k testovaniu	Bc. Lukas Hagara	Done
4547 Task	Otestovať vuzix okuliare	Bc. Marek Rostar	Done
4548 Task	Otestovať web kameru, face detection	Bc. Marek Rostar	Done
4549 Task	Otestovať kinect	Bc. Peter Belai	Done
4550 Task	Otestovať načítanie zo súborov	Bc. Lukas Hagara	Done
4551 Task	Otestovať otáčanie grafov	Bc. Martin Mokry	Done
4552 Task	Otestovať funkčnosť aplikácie	Bc. Lukas Hagara	Done
4556 Task	Opraviť warning	Bc. Patrik Berger	Done
4560 Task	Dokončiť build	Bc. Marek Rostar	Done
4558 Task	Vygenerovať doxygen dokumentáciu	Bc. Filip Skultety	In Progress
4559 Task	Vygenerovať sphinx dokumentáciu	Bc. Filip Skultety	Done

Obrázok 3.2.1 Export úloh 1. šprint z TFS

3.3. Šprint č. 2

Počas druhého šprintu sme sa hlbšie zoznamovali so zdrojovým kódom projektu. Opravovali sa upozornenia z kompilátora, cppchek a cplint, ostatné úlohy sú znázornené na [Obrázok 3.3.1]. Na konci šprintu sme predviedli vlastníčkovi produktu aktuálny stav a výstupy jednotlivých úloh.

Retrospektíva

Počas retrospektívy šprintu č.2 sme spoločne identifikovali nasledovné nedostatky, ktoré sa budeme snažiť v budúcich šprintoch zlepšiť:

- Pri oprave upozornení z kompilátora, cppchek a cpplint dvojice nemali zosynchronizované vetvy. Keďže všetky tri úlohy boli navzájom naviazané, synchronizácia by výrazne zefektívnila overovanie opráv upozornení.
- Počas plánovania úloh sme zabudli ohodnotiť úlohy, čo spôsobilo, že sa nám nepodarilo vygenerovať burndown chart pre šprint 2.

Project: AugReality Server: tfs.fiit.stuba.sk\StudentsProjects Query: Report Sprint 2 List type: Flat		
ID	Work Item Type	Title
4558	Task	Vygenerovať doxygen dokumentáciu
4812	Task	Oprava warningov z kompilátora Windows
4813	Task	Oprava warningov z kompilátora Linux
4814	Task	Oprava warningov cppcheck
4815	Task	Oprava warningov cpplint
4816	Task	3d mys zmena použitia
4820	Task	Analyza kniznic pre podporu K2 s freenect,openni
4822	Task	Ozivit tlačitko kinect snapshot
4823	Task	Opis Kinect use casov
4824	Task	Vytvorit dataset via K1
4840	Task	Zdokumentovat štruktúru grafu sceny
4852	Task	Vytvorenie diagramu tried 1
4853	Task	Vytvorenie diagramu tried 2

Obrázok 3.3.1 Export úloh 2. šprint z TFS

4. Používané metodiky

Počas projektu používame nasledovné metodiky:

- Ako programovať v C++ - Ako písať headre a cpp súbory
- CppCheck – Analýza cpp kódu
- CppLint – c++ style kontrola
- Astyle – formátovanie a kontrola c++ kódu
- CodeReview – Kontrola vytvoreného kódu
- GitFlow – práca s gitom (vetvy, pull requesty, commit správy).
- Tvorba a údržba UML diagramov prostredníctvom PlantUML
- Sphinx – práca s nástrojom sphinx (tvorba html, latex dokumentácie)
- TFS – manažment úloh
- Logovanie s knižnicou EasyLogging++ - logovanie záznamov, chýb, pomocné výpisy
- Písanie BDD testov

5. Globálna retrospektíva ZS

Prvé týždne semestra sme sa oboznamovali s projektom a pripravovali si softvérové prostredie pre ďalší vývoj. Po inštalovaní softvérov sme sa začali riadiť existujúcimi metodikami a chýbajúce metodiky (k riadeniu úloh, cplusplus, cppcheck, style, logovanie, písanie testov) sme vytvorili. Počas semestra sa vyskytli problémy s nedodržiavaním metodík, hlavne s gitflow metodikou.

6. Motivačný dokument

6.1. Tím

Náš tím sa skladá zo 7 ľudí. Sú to Michal Galbavý, Peter Belai, Patrik Berger, Marek

Roštár, Lukáš Hagara, Martin Mokrý a Filip Škultéty. Všetci sme absolventi bakalárskeho štúdia na FIIT STU. Vďaka tomu sme nadobudli znalosti z rôznych technológií, ako je napr. Java, C. Časť tímu má taktiež skúsenosti s funkcionálnym programovaním alebo spracovaním obrazu a počítačovej grafiky. Väčšina členov tímu má taktiež skúsenosti, či už zo školských projektov, bakalárskych prác alebo zo zamestnania, s webovými technológiami HTML, CSS, JavaScript a PHP. Dvaja členovia z tímu majú taktiež skúsenosti s webovým frameworkom django. Ako tím máme široký záber rôznych technológií, ktoré sme nadobudli pri štúdiu alebo pri práci, takže členovia tímu ovládajú mnoho rozličných technológií. Medzi ne patria napr. jazyky python, R, C++, Lisp, Prolog, RESTové služby, XML, XSLT, UML, XPath a framework Spring. Na inžinierskom štúdiu sme si, k nami najviac preferovaným témam, zvolili predmety: vyhľadávanie informácií, vizualizácia dát a objavovanie znalostí.

6.2. Motivácia - Extrakcia dát z webu [WebExtraction]

Túto tému sme si zvolili ako prioritnú z viacerých dôvodov. Jedným z nich je možnosť preukázania nášho riešenia nad štátnymi dátami. Tie sú pre náš tím zaujímavé, pretože sa vďaka nim dá odhaliť mnoho rôznych súvislostí. S tým má už jeden člen tímu, Peter Belai, skúsenosti. Tie nadobudol pri hackatone, ktorý bol zameraný práve na štátne dáta. Tento hackaton taktiež vyhral. Ďalšou skúsenosťou je práca pre Alianciu Fair Play, kde extrahoval štrukturované dáta zo štátnych webov. Okrem týchto skúseností je pre nás motivácia naučiť sa pracovať s cloudovými technológiami. V neposlednej rade by sme radi pracovali na tejto téme z dôvodu dobrých recenzií na vedúceho pre danú tému. Do projektu by sme mohli priniesť viaceré skúsenosti. Časť tímu nadobudla skúsenosti s XPathom a XML schémou, či už z predmetu webové publikovanie alebo z pracovných skúseností, kde sa upravovali XML súbory v docx súboroch. Mimo iného má veľká časť tímu skúsenosti s webovými technológiami. Tieto sme nadobudli tiež z práce, kde sme vytvárali webové stránky, napr. aj e-Shopy, alebo aj zo školských projektov. S ruby on rails sme sa doposiaľ nestretli, ale sme otvorení novým príležitostiam naučiť sa túto technológiu

6.3. Motivácia - Inteligentný sklad [SmartStore]

Jedna z charakteristických vlastností tejto témy, ktorá nás zaujala je práca so skutočným zákazníkom. Komunikácia s ním, spracovanie požiadaviek a možnosť voľne zvoliť webové technológie je to čo nás primárne oslovilo. Taktiež vidíme možnosť využiť metódy strojového učenia, ktoré sú z nášho pohľadu veľmi zaujímavou oblasťou informatiky.

Členovia nášho tímu majú skúsenosti s tvorbou rozsiahlych webových aplikácií, vytváraním e-shopov a v rámci bakalárskych prác sa vo väčšom rozsahu venovali genetickým algoritmom, heuristikám a klasifikácii. Niektorí členovia majú zapísaný predmet Objavovanie znalostí, ktorý by mohol byť prínosom pri riešení tejto témy. Po krátkom brainstormingu sme identifikovali ďalšie vlastnosti tovaru, ktoré by bolo vhodné sledovať:

- rok vydania tovaru (nové vydanie, staré tituly),
- úspešnosť autora knihy, na základe predchádzajúcich diel,
- žáner knihy,
- zľava na daný tovar,
- predajnosť cez sviatky, víkendy, prac. dní a ročné obdobia,
- udalosti v živote autora.

Ako sme vyššie spomínali systém by fungoval s využitím strojového učenia. Toto strojové učenie by sme obohatili o rôzne heuristiky. Tieto by mohli byť následne upravované genetickým algoritmom, ktorý by prispôboval hodnoty parametrov. V riešení by sme chceli optimalizovať inteligentný sklad v 3 úrovniach:

- Tovar v sklade (optimálny počet produktov v sklade)
- Efektívne doplnenie tovarov do jedného skladu(správne naplánovať počet kusov tovaru, dátum, využitie dodávacieho vozidla a pod.)
- Efektívne doplnenie tovarov do viacerých skladov

6.4. Motivácia - Vizualizácia informácií v obohatenej realite [AugReality]

Táto téma sa radí medzi naše prioritné z viacerých dôvodov. Zaujímá nás uľahčovanie hľadania súvislostí s využitím rozličných metod vizualizácie. Túto tému vnímame ako výzvu, ktorá spočíva v práci s neprebádanými technológiami. Veríme v potenciál objavenia nových prístupov pri práci s vizualizačnými technológiami. Časť nášho tímu bola motivovaná k výberu tejto témy prednáškou docenta Kapca o vizualizácií na predmete ICP. Viacerí členovia absolvovali počas bakalárskeho štúdia predmet PPGSO a tak majú skúsenosti s programovacím jazykom C++ a knižnicou OpenGL. Ďalší z členov pracoval pri bakalárskej práci s komerčnými senzormi. Následne máme členov, ktorí majú skúsenosti s alternatívnymi spôsobmi ovládania počítača. Týmito spôsobmi sú konkrétne ovládanie hlasom a myslou. Veríme, že by sme ovládanie gestami mohli týmto spôsobom doplniť a potenciálne aj vylepšiť

Poradie nami zvolených tém:

1. WebExtraction
2. SmartStore
3. AugReality
4. EduSim
5. CodeCrutches
6. DronSim V2
7. Story Teller
8. DeepSearch
9. VirtualFEI
10. 3D futbal

7. Metodiky

V projekte nadväzujeme na prácu predchádzajúcich študentov, preto sme niektoré metodiky prebrali z minulých rokov a niektoré sme si vytvorili alebo čiastočne upravili pre tohtoročné potreby.

7.1. Ako programovať v C++

Ako písať headre a cpp súbory

Includuje sa v tomto poradí (platí pre .h aj .cpp):

- headre z projektu
- headre z Qt a OSG
- systémové headre

v .cpp je prvý header prislusne .h-cko toho .cpp

Important: pravidlo: includujem najprv tie, ktoré môžu includnut čo najviac

Pravidla pre písanie .h

- obsahujú iba definície metod, NIE implementáciu metod
- includujeme LEN čo je treba pre header, NIC naviac
- ak je v triede/metode použitý pointer typ, napr. Node* n;

tak stačí použiť doprednú deklaráciu, t.j. class Node;

a netreba includovať Node.h

Attention: toto nefunguje, keď:

- sa dedí trieda
- to nie je pointer, t.j. je to Node n;
- keď je použitý osg::ref_ptr pointer, resp. templaty

Pravidla pre písanie .cpp

ak je typ premennej použitý LEN v .cpp (typicky lokálna premenná v metode), tak príslušný header dávame len do .cpp

ZLE praktiky, resp. čo nerobiť

NIKDY v headroch a cpp súboroch nepoužívať “using namespace”

- nepoužívať “using” keyword

ak už je použité, tak to treba upraviť na :

```
namespace XYZ {  
... implementacia ...  
}
```

NEPOUŽÍVAŤ "0" (nulu) ako NULL pointer, ale

- použiť nullptr (ak kompilator podporuje C++11), prípadne použiť NULL

DOBRE praktiky, resp. čo robiť

inicializovať VŠETKY atributy "pri" konštruktore cez "initialization list"

- najme pointre

– skontrolovať ak je "new" v konštruktore, tak MUSÍ byť "delete" v deštruktore (neplatí pre: `osg::ref_ptr`)

- inicializovať v takom poradí v akom sú zapísané v triede:

– usporiadať atributy od "najväčších" (napr. pointer, trieda, double) po "najmenšie" (int, char, bool)

aj keď sa bude mixovať public/private

"std::cout"

- pokiaľ je v kóde, ktorý používa Qt, tak prerobiť na qDebug
- resp. nájsť a používať nejakú externú knižnicu na logovanie

Ak sa použije cudzí existujúci kód tak, že sa priamo jeho zdrojáky pridajú do existujúceho projektu

- tak treba aby zostal pôvodný a úpravy sa riešili napr. v zdedenej triede

WARNING-y - opravujú sa všetky warningy v našom kóde (minimalizácia možných problémov)

pravidelne sa robí statická analýza kódu pomocou nástrojov cppcheck a cplint

- vid. metodiky cppcheck, cplint

pravidelne sa zdrojový kód formátuje pomocou nástroja astyle pravidelne sa mergejú všetky práce.

7.2. CppCheck

<http://cppcheck.sourceforge.net>

Použite cppcheck v 1.70

Ako používať

1. v Edit->Preferences nastaviť: General: check "Force checking all #ifdef configurations" Paths: pridať cestu do "RealityNotFound/include" Advanced: check "Show inconclusive errors" !!! toto hlási dosť false-positive, ALE občas nájde dôležité veci !!!

2. Check->Directory a vybrať "RealityNotFound/src"

poznámky k reportom: !!! ak je niečo nejasne, treba sa opýtať !!!

!!! ignorovať hlášky pre externý kód: !!! noiseutils, qtcolorpicker

Dôležité hlášky typu:

Technically the member function XYZ can be const.

- najmä funkcie typu "getter" môžu byť const, napr.:

```
int getX () const {  
    return 1;  
}
```

Technically the member variable XYZ can be const.

- treba skontrolovať!!!

The class 'RestrictionsManager' does not have a constructor.

- každá trieda by mala mať konštruktor (kompilér nám síce vytvorí default, ale... vid. nižšie)

'class Type' does not have a copy constructor which is recommended since the class contains a pointer to allocated memory.

- závisí, či sa "copy constructor" v kóde používa - nutná kontrola

The scope of the variable 'gesto_hore' can be reduced. Variable 'gesto_hore' is assigned a value that is never used.

- to je jasné

C-style pointer casting

- to je riešene aj cez cplint v samostatnej hotfix branch
- v gcc da sa zapnúť `-Wold-style-cast` -> momentálne hlási veľmi veľa warningov

- je to pre čitateľnosť, ALE ma to svoje opodstatnenie

Member variable 'Cube::at' is not initialized in the constructor.

- všetky class variables by mali byť inicializovane v konštruktoze cez "initialization list"

!!! najmä pointre !!! na NULL, resp. cez new (a delete v destructore)

Possible null pointer dereference: conn - otherwise it is redundant to check it against null. Possible leak in public function. The pointer 'nodeTypeComboBox' is not deallocated before it is allocated.

!!! treba skontrolovať - indikuje závažnú chybu !!!

Uninitialized variable: newGraph

!!! treba skontrolovať - indikuje závažnú chybu !!!

ostatne hlasky

- treba skontrolovať - a mali by sa opraviť \

7.3. CppLint

Použitý cplint dostupny:

<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CB8QFjAA&url=http%3A%2F%2F>

styleguide.googlecode.com%2Fsvn%2Ftrunk%2Fcpplint%2Fcpplint.py&ei=li6pVO_bK8mAUZSVgSg&u sg=AFQjCNGnSkFFrrX3TI Eh8vRmUDJQ&bvm=bv.82001339,d.d24

Je to Python script.

Ako používať (Linux)

pouzite testy su nastavene v CPPLINT.cfg dostupne testy sa vypisu: `cpplint.py -filter=`

spustit v include adresari: `cpplint.py Aruco/* Core/* Data/* Importer/* Layout/* Math/* Network/* Network/executors/* OpenCV/* QOpenCV/* Speech/* Viewer/* Clustering/* Clustering/Figures/* Fglove/* Kinect/* Kinect/RansacSurface/* Manager/* Model/* OsgQtBrowser/* QOSG/* Util/*`

2>&1 | tee report-include.txt

spustit v src adresari: cpplint.py Aruco/* Core/* Data/* Importer/* Layout/* Math/* Network/* Network/executors/* OpenCV/* QOpenCV/* Speech/* Viewer/* Clustering/* Clustering/Figures/* Fglove/* Kinect/* Kinect/RansacSurface/* Manager/* Model/* OsgQtBrowser/* QOSG/* Util/*

2>&1 | tee report-src.txt

Poznamky k reportom

!!! ak je niečo nejasne, treba sa opytat !!!

ignorovat hlášky pre externy kod: noiseutils, qtcolorpicker ciastocne pre cameramanipulator, QGraphicsViewAdapter

dolezite hlášky typu:

Constructors callable with one argument should be marked explicit

- nastudovat, asi staci pridať keyword explicit

Is this a non-const reference? If so, make const or use a pointer

- nastudovat (ci to nieje false-positive)

Use int16/int64/etc, rather than the C type long

- nastudovat

Do not use namespace using-directives.

- odstranit "using namespace" (okrem externeho/cudieho kódu) Consider using rand_r(...) instead of rand(...) for improved thread safety.

- keďže máme vlákna, asi by bolo vhodné. vid.:

<https://stackoverflow.com/questions/3973665/how-do-i-use-rand-r-and-how-do-i-use-it-in-a-thread-safe-way>

Are you taking an address of a cast? This is dangerous: could be a temp var. Take the address before doing the cast, rather than after

!!! indikuje vazny problem !!!

false-positive:

Clustering/Figures/Cube.h:28: Add #include <algorithm> for transform
[build/include_what_you_use] [4]

Cube ma metodu tranform -> netreba include

3.2.2 ostatne hlasky:

- treba skontrolovať - a mali by sa opraviť (resp. už som ich opravil ;-)

7.4. PlantUML

PlantUML je jednoduchý program na tvorbu UML diagramov prostredníctvom ich textového opisu. K samotnému programu prislúcha aj rozsiahla dokumentácia.

PlantUML je voľne dostupný na [stiahnutie](#) z oficiálnej stránky, prípadne je možné na otestovanie použiť aj jednoduchú [web aplikáciu](#).

Pre plnohodnotné využitie je potrebné mať taktiež nainštalovaný [Graphviz](#).

Tiež ponúka možnosť [integrácie](#) s viacerými textovými editormi a wiki stránkami.

Pravidlá pre tvorbu súborov

1. Každý diagram sa nachádza v samostatnom textovom súbore (koncovka .txt, resp .wsd pri použití integrácie so sublime text).
2. Vygenerovaný diagram má identický názov ako prislúchajúci textový súbor (koncovka .png).
3. Názvy súborov sú po anglicky.

Odstránenie duplicity pomocou Preprocesoru

Pri písaní diagramov, ktoré obsahujú komplikované vzťahy medzi entitami môžeme naraziť na situáciu, kde budeme veľa krát za sebou písať ten istý názov triedy alebo metódy. S využitím makier preprocesoru môžeme túto duplicitu ľahko odstrániť.

```
@startuml
'Bez preprocesoru
package "class Filter representation" {
class ObjectStructure class Element {
+{abstract}register(Visitor v)
}
class Mapper {
    +register(Filter f)
}
class Klient class Visitor {
    +{abstract}visitMapper(Mapper m)
}
```

```
class Filter {  
    +visitMapper(Mapper m)  
}
```

ObjectStructure -down-> Element

Mapper -up-|> Element

ObjectStructure <-left- Klient

Klient -down-> Visitor

Filter -up-|> Visitor

```
}
```

```
@enduml
```

```
@startuml
```

```
'S preprocesorom
```

```
!define o(x) ObjectStructure
```

```
!define e(x) Element
```

```
!define m(x) Mapper
```

```
!define k(x) Klient
```

```
!define v(x) Visitor
```

```
!define f(x) Filter
```

```
package "class Filter representation" {
```

```
class o()
```

```
class e() {
```

```
+{abstract}register(v(x) v)
```

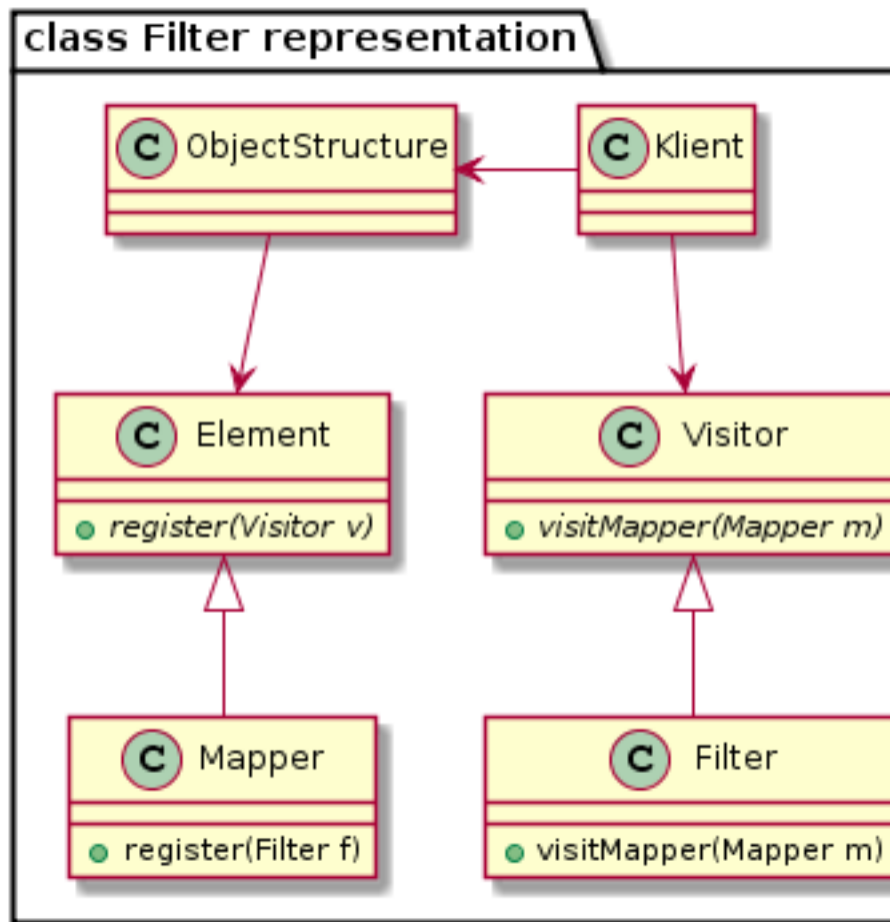
```
class k()
```

```
class v() {
```

```
+{abstract}visitMapper(m(x) m)
```

```
}  
class f() {  
+visitMapper(m(x) m)  
}  
o() -down-> e()  
m() -up-|> e()  
o() <-left- k()  
k() -down-> v()  
f() -up-|> v()  
}  
@enduml
```

V oboch prípadoch bude výsledok nasledovný:



V druhom prípade sa rozhodne menej napíšeme a máme možnosť meniť použité názvy tried na jednom mieste namiesto toho aby sme ich museli meniť všade. Stojí za poznámku, že každé definované makro musí mať parameter (v našom prípade x, z ktorého ajtak nečítame). Viac o Preprocesore na tejto stránke.

Použitie aliasov v sekvenčnom diagrame

V sekvenčných diagramoch odporúčame pri definovaní volaní medzi objektami používať aliasy (skratky). Ich princíp je analogický s predchádzajúcim makrom avšak sú ešte o niečo prehľadnejšie. Aliasy nie sú však podporované v class diagrame.

@startuml

participant Client as c participant Server as s

title sd Basic Server call activate c

c -> s: sendMessage("You are awesome!")

activate s

s --> c: result = "ok" deactivate s

deactivate c

@enduml

Výsledok:



Užitočnosť týchto skratiek (a makier) pochopiteľne narastá s narastajúcou komplexnosťou daného diagramu.

Pravidlá pre súborovú štruktúru

Samotné UML diagramy je potrebné rozdeliť do prehľadnej súborovej štruktúry:

- projekt (názov projektu, napr. 3dsoftviz)
 - doc (inštalačná dokumentácia, vygenerovaná dokumentácia atď.)
 - uml
 - * structural
 - class diagrams (korešpondujúce s reálnym kódom)

- component diagrams

* behavioral

- activity diagrams
- use-case diagrams
- sequence diagrams
- state diagrams

7.5. GitFlow

Vetvy

- Master - hlavný projekt
- Develop - branchnutá z mastra, každý šprint má vlastnú Develop vetvu, na konci šprintu sa merge späť do mastra
- Feature - branchnutá z developu, každý nový kus funkcionality (task v TFS), ktorý sa kódi musí mať vlastnú

– Feature vetvu... po dokončení a validácii kódu sa merge späť do Developu, NEINTERAGUJE S MASTER VETVOU

- Hotfix - vetva na rýchly fix priamo z mastera, merge sa do mastera AJ developu, navyšuje aktuálnu verziu

Vždy mergejeme cez Shell a s prepínacom –no-ff

Cheat sheet so všetkými základnými príkazmi:

<https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf>

Pull requesty

Po dokončení práce, keď sme ready to review sa dáva pull request na vetvu, do ktorej sa bude mergevať.

Pull request sa robí z GUI GitHubu (pravý horný roh), alebo \$ git request-pull {meno_commitu} {URL} (doporučujem robiť cez GUI)

Po odsúhlasení Pull requestu sa potom pristúpi k merge.

Commit messages

v Commit messages používame tagy na začiatok:

- [FIX] - fixli sme nejakú chybu z minula, bugfix, hotfix a podobne
- [ADD] - pridali sme novu funkcionalitu, subor, ...
- [DOC] - pridali sme dokumentáciu, komentý...
- [REF] - pre refactoring
- [FMT] - formátovanie textu, úprava
- [TEST] - pre testy

Za tým veľmi stručne (a výstižne) opíšeme, aké zmeny sme spravili. Message by mali byť krátke, no pokrývať všetko, čo sme v commite spravili.

Useful commands

- `$ git submodule update --init --recursive`
– update submodulov (dependencies)
- `$ git checkout -f meno_branch`
– checkout branche aj napriek lokálnym zmenám, budú zahodené
- `$ git status`
– vypíše všetky vykonané zmeny
- `$ git stash / $ git stash pop`
– uloží stav projektu do stashu, z ktorého sa dá potom tento stav pop-núť, dobre na prenos zmien medzi vetvami

Tvorba feature branch-u

- `$ git checkout -b "feature/meno-feature" develop //Switched to a new branch "feature/meno-feature"`

Mergovanie hotového feature:

- `$ git checkout develop //Switched to branch 'develop'`
- `$ git merge --no-ff meno-feature`
- `$ git push origin develop`

Tvorba hotfix branch-u:

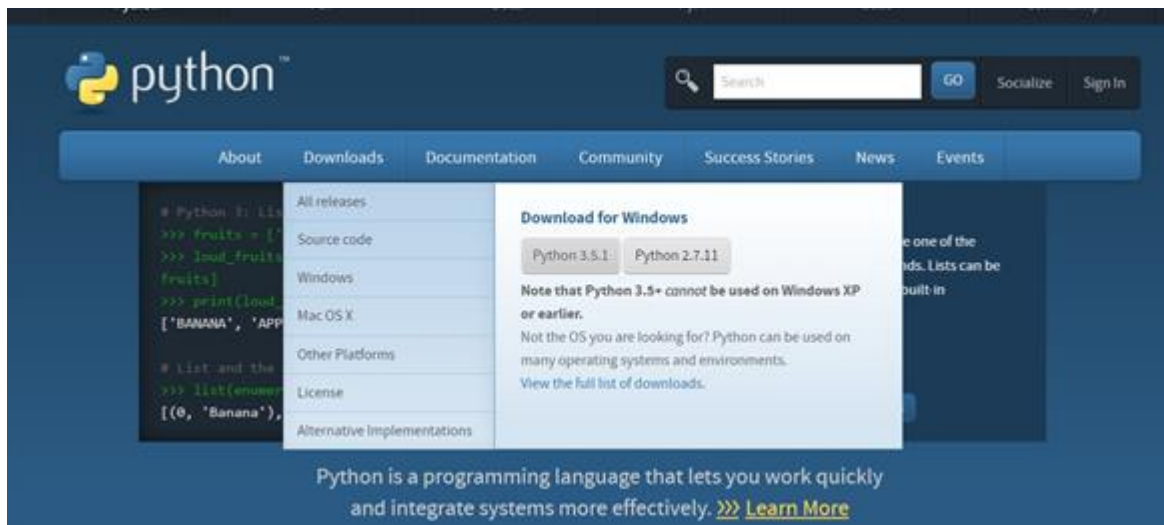
- `$ git checkout -b "hotfix/nazov-co-fixujem" master //Switched to a new branch "hotfix-{cislo_verzie}"`
- `$ git commit -m "sprava, čo som spravil"`

Uzatvorenie Hotfix branchu:

- `$ git checkout develop //Switched to branch 'develop'`
- `$ git merge --no-ff "hotfix/nazov-co-fixujem"`

7.6. Sphinx dokumentácia

Pre prácu so Sphinxom treba mať nainštalovaný Python.



Python ponuka verzie 2.x a 3.x. Sphinx 1.3 môže bežať pod Python 2.6, 2.7, 3.3, 3.4, ale odporúčaná verzia je 2.7.

Pre stiahnutie a inštalovanie externých knižníc pre Python existuje príkaz pip. Príkaz už sa nachádza v oficiálnych verziách Pythonu 3.4.0 alebo 2.7.9.

Ak sa príkaz nenainštaloval automaticky, treba ho stiahnuť zo stránky <https://bootstrap.pypa.io/get-pip.py> a niekam uložiť. V príkazovom riadku treba prejsť do adresára s get-pip.py a spustiť nasledovný príkaz:

```
python get-pip.py
```

Sphinx

- Prejsť do priečinku s dokumentáciou (tam kde index.rst sa nachádza) a pomocou príkazu pip nainštalovať Sphinx:

```
pip install sphinx
```

- Ak treba vytvoriť novú dokumentáciu, pre nastavenie zdrojového adresára a vytvorenie potrebných súborov na prácu so Sphinx treba spustiť príkaz:

```
sphinx-quickstart
```

a odpovedať na otázky. Vyberte si všetky predvolené odpovede a po vyzve zadajte názov, autorov a verziu projektu.

- Týmto príkazom budú vygenerované súbory Makefile, make.bat a conf.py.in.

– Všetky konfigurácie dokumentácie sú v conf.py.in.

Upozornenie: Sphinx-quickstart a vytváranie týchto súborov generujú novú dokumentáciu! Ak súbory index.rst, Makefile, make.bat a conf.py.in už existovali, tak sa prepisu!

- Sphinx dokumentácia generuje výstup v rôznych formátoch zo súborov .rst.

HTML dokumentácia

- Súbor make.bat povolí vygenerovať dokumentáciu v tom formáte, ktorý potrebujete
- Pre generovanie HTML dokumentácie treba v príkazovom riadku prejsť do priečinku s ReST súbormi a make.bat súborom a spustiť príkaz

```
make html
```

- Inak generovanie dokumentácie sa dá spustiť pomocou CMake v QtCrateore.

PDF dokumentácia

Pre generovanie PDF dokumentácie potrebujeme najprv vytvoriť Latex dokumentáciu.

Note: Pre prácu s Latex treba mať TeXlive.

Príkazom

```
make latex
```

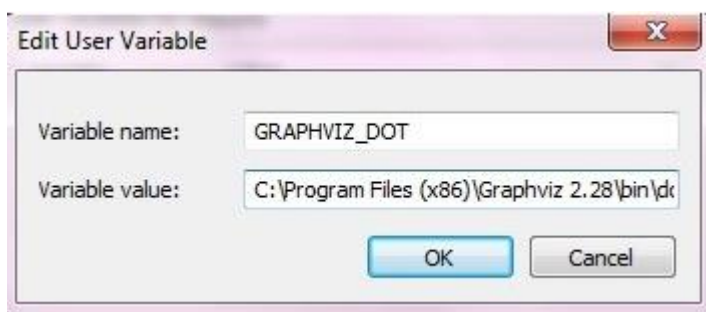
vygeneruje sa Latex dokumentácia, ktorá následne sa môže konvertovať do PDF pomocou programu TeXstudio.

PDF dokumentacia generuje len pomocou prikazoveho riadku a externeho programu, neda sa spustat cez CMake!

PlantUML

Pre prácu s PlantUML nastrojmi v Sphinx treba:

- nainstalovat Javu
 - pridať Javu do premennych prostredi (environment variable)
 - nainstalovat Graphviz
- odporúčaná verzia je 2.28
- pridať Graphviz do premennych. Hodnota premennej ma byt do dot.exe:



- pridať Graphviz do extensions v conf.py.in:

```
extensions = ['sphinx.ext.graphviz']
```
 - nainstalovat sphinxcontrib-plantuml zo stranky alebo prikazom

```
pip install sphinxcontrib-plantuml
```
 - pridať plantuml do extensions v conf.py.in:

```
extensions = ['sphinxcontrib.plantuml']
```
 - stiahnut plantuml.jar
 - pridať do conf.py.in prikaz:

```
plantuml = 'java -jar /cesta/do/plantuml.jar'
```
- Attention: Dolezite je zmenit tuto cestu na spravnu, aku mate aktualnu na Vasom pocitaci!
- pridavat UML do dokumentacii je mozne pomocou

.. uml::

linclude /cesta/do/subor.wsd(txt)

alebo

@startuml

PlantUML kod

@enduml

7.7. TFS

Metodika, ktorá určuje pravidlá a postupy na manažment úloh v nástroji TFS.

Pridanie novej úlohy - všeobecne

- Zadať ohodnotenie úlohy, priority, opis, Zadať keď je úloha hotová.
- Zadať odhadovaný čas dokončenia.
- Opis musí byť podrobný, aby každému členovi bolo jasné, čo má vykonať po pridelení úlohy.

Rozdeľovanie úloh

- Každý si vyberie (potiahne) úlohu/úlohy, ktorá/é majú najvyššiu prioritu.
- Ak ostanú nepridelené úlohy, študentský vedúci tímu pridelí členom zvyšné úlohy.

TFS

- Adresa TFS: <https://tfs.fiiit.stuba.sk:8443/tfs/> (Potrebné sa prihlásiť 2x)
- Projekt: AugReality / AugReality Team

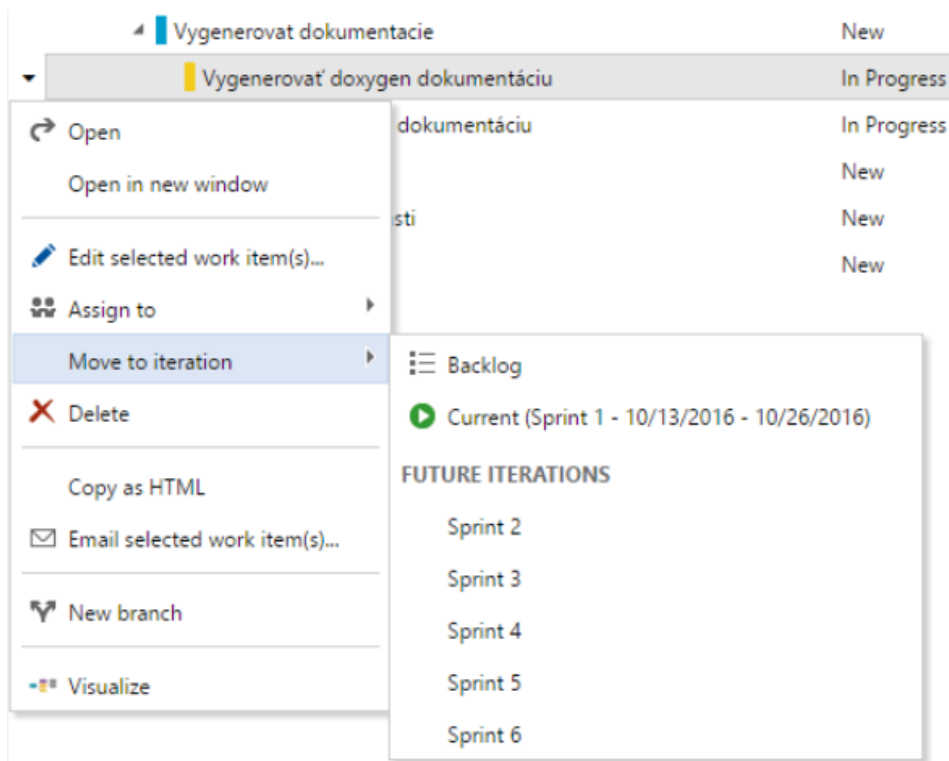
Pridanie nových úloh – TFS

- Úlohy podobného charakteru priradíme do spoločného backlog itemu.
- Pri pridávaní úlohy sa automaticky nastaví stav 'To Do'.
- V prípade objavenia chyby, je potrebné vytvoriť novú úlohu typu Bug (Chyba)

Úlohy (tasky) sa môžu nachádzať v troch stavoch:

- To Do
- In Progress
- Done

Nesplnené úlohy, ktoré sa nestihli dokončiť v danom šprinte, presunieme do nasledujúceho šprintu.



Obrázok 7.7.1 Presun úlohy do ďalšieho šprintu

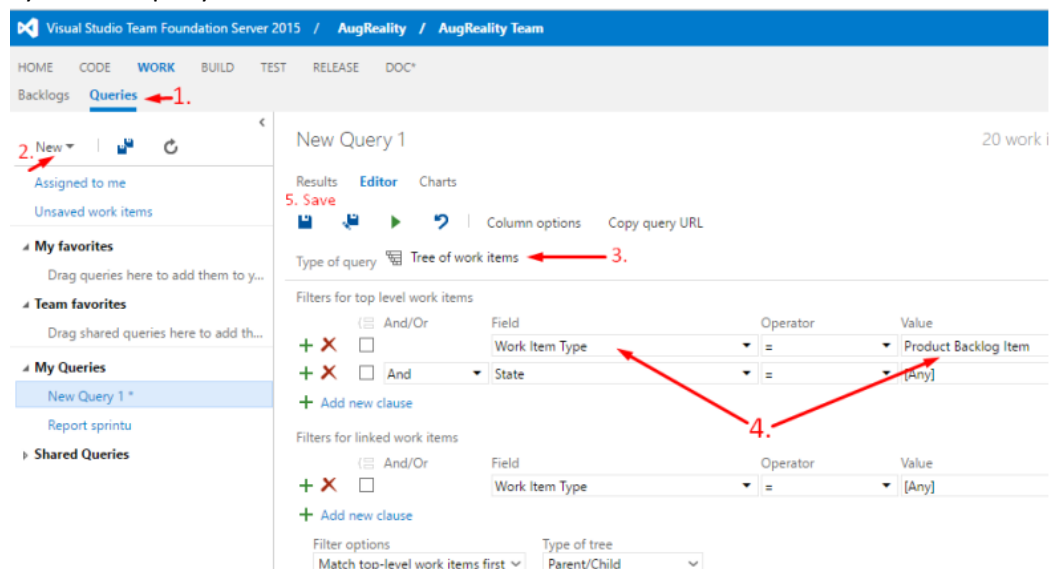
Vytváranie exportov z TFS

Navod na stránke: [https://msdn.microsoft.com/en-us/library/dd286627\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd286627(v=vs.110).aspx)

URL: <https://tfs.fiit.stuba.sk:8443/tfs/>

Login: ako do AIS-u

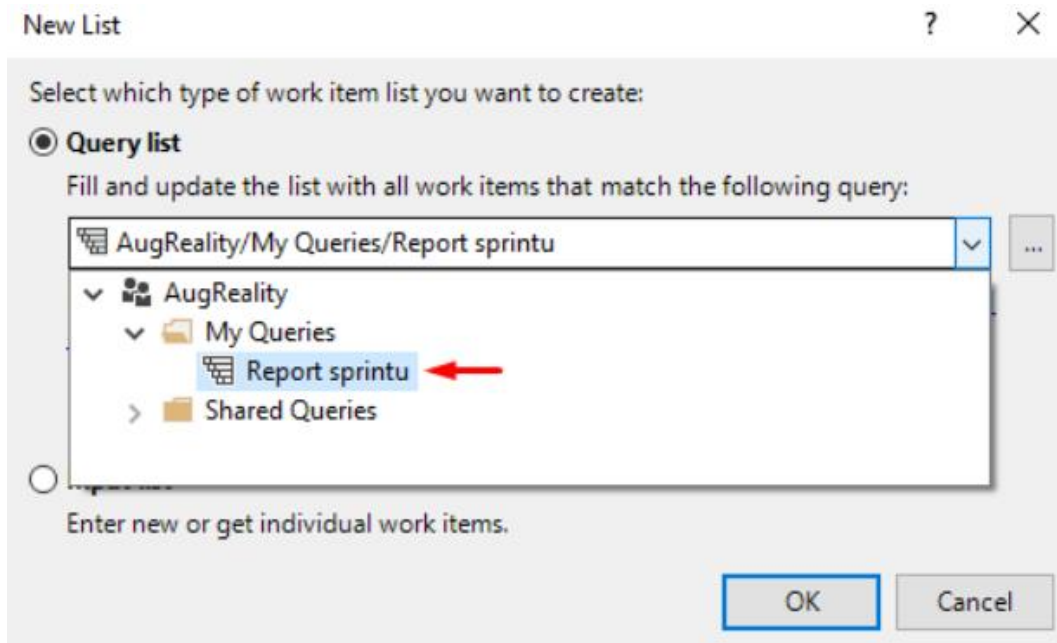
Pred prvým exportom je potrebné si vytvoriť query, ktoré vráti stav úloh v danom šprinte. Návod na vytvorenie query:



Obrázok 7.7.2 Vytvorenie query v TFS

Import do Excelu

- V hlavnom menu vybrať záložku TEAM (mala by byť vpravo hore).
- Klik na New List (umiestnená na ľavo pod záložkou File/Súbor).
- Vybrať novo vytvorené query:



Obrázok 7.7.3 Výber query v excely

7.8. Logovanie s knižnicou Easylogging++

Úrovne logovania

Easylogging++ poskytuje tieto úrovne logovania:

- Global - Generická úroveň, reprezentujúca všetky ostatné
- Trace - Zachytávame informácie, vhodné na back-trackovanie rôznych problémov.
- Debug - Zaznamenávame informácie, vhodné pri vývoji aplikácie.
- Fatal - Používame, keď nastane chyba, ktorá pravdepodobne ukončí program.
- Error - Používame, keď nastane závažnejšia chyba ale program bude naďalej pracovať.
- Warning - Používame keď nastane chyba ale program bude naďalej pracovať.
- Info - Používame na zachytenie priebehu aplikácie.
- Verbose - Nepoužívame.
- Unknown - Nepoužívame.

Čo a ako treba logovať

- Začiatok každej metódy
- Všetky možné error / warningy

Ako treba logovať:

- Začiatok každej metódy logujeme v tvare :

```
function(){  
LOG( INFO ) << "MENO_BALIKU/MENO_TRIEDY/MENO_FUNKCIE(PARAMETRE)"  
}
```

Pozor, parametre používať iba v prípade, že je to vhodné. Všetky možné error / warningy logujeme v tvare:

```
LOG( WARNING/ERROR/FATAL ) << "MENO_BALIKU/MENO_TRIEDY/MENO_  
->FUNKCIE(PARAMETRE)"
```

V tomto prípade sa snažíme zalogovať aj všetky potrebné parametre, ktoré spôsobili warning / error.

7.9. Metodika na písanie BDD testov

1. Vytvorme normálny .cpp súbor a v hlavičke uvedieme:

```
#include <igloo/igloo_alt.h>  
  
using namespace igloo;
```

2. Najskôr popíšeme, čo chceme testovať pomocou Describe-u:

```
Describe(a_foo_bar) {  
...  
}
```

3. V tele Describe-u uvedieme funkcie, ktorými budeme testovať správanie. Funkcie pomenujme tak, aby boli samoopisné:

```
It( foo_should_be_bar ) {  
...  
}
```

4. V tele It-u už píšeme klasické asserty, napr:

```
Assert::That( 2+4, Is().EqualTo(6) );  
Assert::That( "FIIT", Is().EqualTo("LIFE") );
```

Spustenie testov

1. Testy (všetky *.cpp) pre váš modul umiestňujte do priečinka:

```
~root/tests/Foo
```


Testy fungujú tak, že si buildnete vlastný .exe súbor, ktorý otestuje danú funkcionality. Test si buildnete nasledovne:

- otvorte hlavný CMakeLists.txt
- ctrl+f -> "BDD Igloo tests"

2. Teraz si potrebujete nakopírovať nejaký kód, minimálne by ste mali mať:

```
file( GLOB_RECURSE SRC_FOO_TESTS "tests/Foo/*.cpp" )
add_executable( test_foo_module ${TEST_RUNNER} ${SRC_FOO_TESTS} )
add_dependencies( test_foo_module igloo )
target_compile_options( test_foo_module PUBLIC ${FLAGS_FOR_DEBUG} )
```

Dole pod testami (alebo ctrl+f -> "run all tests") pridajte ešte jeden riadok kódu:

```
add_test( testing_foo_module test_foo_module )
```

3. Teraz musíte zmeny uložiť a nechať zbehnúť CMake

- pravým na project -> Run CMake
- v QtCreatori zmeňte build/run target na Tests/test_foo_module

4. Pokiaľ už máte v priečinku tests/foo testy mali by sa buildnúť/zbehnúť

8. Export evidencie úloh

V kapitole sú zobrazené exporty úloh.

8.1. Šprint č. 1, týždeň 1

Project: AugReality Server: tfs.fiiit.stuba.sk\StudentsProjects Query: Report sprintu List type: Tree						
ID	Work Item Type	Title 1	Title 2	Assigned To	State	Tags
4539	Product Backlog Item	Sprava metodik			New	
4540	Task		Metodika ku kvalite	Bc. Martin Mokry	Done	
4541	Task		Metodika gitflow	Bc. Patrik Berger	Done	
4542	Task		Metodika k TFS	Bc. Filip Skultety	Done	
4543	Task		Metodika ku code review	Bc. Patrik Berger	Done	
4544	Task		Metodika k logovaniu	Bc. Peter Belai	Done	
4545	Task		Metodika k testovaniu		To Do	
4546	Product Backlog Item	Otestovať existujúce casti			New	
4547	Task		Otestovať vuzix okuliare	Bc. Marek Rostar	To Do	
4548	Task		Otestovať web kameru, face detection	Bc. Marek Rostar	Done	
4549	Task		Otestovať kinect	Bc. Peter Belai	To Do	
4550	Task		Otestovať načítanie zo súborov	Bc. Lukas Hagara	In Progress	
4551	Task		Otestovať otáčanie grafov	Bc. Martin Mokry	To Do	
4552	Task		Otestovať funkčnosť aplikácie	Bc. Lukas Hagara	In Progress	
4555	Product Backlog Item	Fixnut problemy			New	
4556	Task		Opravit warning	Bc. Patrik Berger	Done	
4560	Task		Dokončiť build	Bc. Marek Rostar	Done	
4557	Product Backlog Item	Vygenerovať dokumentacie			New	
4558	Task		Vygenerovať doxygen dokumentáciu	Bc. Filip Skultety	To Do	
4559	Task		Vygenerovať sphinx dokumentáciu	Bc. Filip Skultety	In Progress	

Obrázok 8.1.1 Export úloh šprint 1 týždeň 1

8.2. Šprint č. 1, týždeň 2

Project: AugReality Server: tfs.fiiit.stuba.sk\StudentsProjects Query: Report sprintu List type: Tree					
ID	Work Item Type	Title 1	Title 2	Assigned To	State
4540	Task		Metodika ku kvalite	Bc. Martin Mokry	Done
4541	Task		Metodika gitflow	Bc. Patrik Berger	Done
4542	Task		Metodika k TFS	Bc. Filip Skultety	Done
4543	Task		Metodika ku code review	Bc. Patrik Berger	Done
4544	Task		Metodika k logovaniu	Bc. Peter Belai	Done
4545	Task		Metodika k testovaniu	Bc. Lukas Hagara	Done
4547	Task		Otestovať vuzix okuliare	Bc. Marek Rostar	Done
4548	Task		Otestovať web kameru, face detection	Bc. Marek Rostar	Done
4549	Task		Otestovať kinect	Bc. Peter Belai	Done
4550	Task		Otestovať načítanie zo súborov	Bc. Lukas Hagara	Done
4551	Task		Otestovať otáčanie grafov	Bc. Martin Mokry	Done
4552	Task		Otestovať funkčnosť aplikácie	Bc. Lukas Hagara	Done
4556	Task		Opraviť warning	Bc. Patrik Berger	Done
4560	Task		Dokončiť build	Bc. Marek Rostar	Done
4558	Task		Vygenerovať doxygen dokumentáciu	Bc. Filip Skultety	In Progress
4559	Task		Vygenerovať sphinx dokumentáciu	Bc. Filip Skultety	Done

Obrázok 8.2.1 Export úloh šprint 1 týždeň 2

8.3. Šprint č. 2, týždeň 1

Project: AugReality Server: tfs.fiiit.stuba.sk\StudentsProjects Query: Report Sprint 2 List type: Flat					
ID	Work	Title	Assigned To	State	Tags
4558	Task	Vygenerovať doxygen dokumentáciu	Bc. Filip Skultety	Done	
4812	Task	Oprava warningov z kompilatora Windows	Bc. Patrik Berger	In Progress	
4813	Task	Oprava warningov z kompilatora Linux	Bc. Marek Rostar	In Progress	
4814	Task	Oprava warningov cppcheck	Bc. Filip Skultety	In Progress	
4815	Task	Oprava warningov cpplint	Bc. Lukas Hagara	In Progress	
4816	Task	3d mys zmena použitia	Bc. Marek Rostar	Done	
4820	Task	Analyza kniznic pre podporu K2 s freenect, openni	Bc. Peter Belai	In Progress	
4822	Task	Ozivit tlačitko kinect snapshot	Bc. Peter Belai	Done	
4823	Task	Opis Kinect use casov	Bc. Lukas Hagara	Done	
4824	Task	Vytvorit dataset via K1	Bc. Peter Belai	Done	
4840	Task	Zdokumentovat štruktúru grafu sceny	Bc. Martin Mokry	Done	
4852	Task	Vytvorenie diagramu tried 1	Bc. Patrik Berger	Done	
4853	Task	Vytvorenie diagramu tried 2	Bc. Martin Mokry	Done	

Obrázok 8.3.1 Export úloh šprint 2 týždeň 1

8.3.1. Šprint č. 2, týždeň 2

Project: AugReality Server: tfs.fiit.stuba.sk\StudentsProjects Query: Report Sprint 2 List type: Flat				
ID	Work	Title	Assigned To	State
4814	Task	Oprava warningov cppcheck	Bc. Filip Skultety	In Progress
4558	Task	Vygenerovať doxygen dokumentáciu	Bc. Filip Skultety	Done
4812	Task	Oprava warningov z kompilatora Windows	Bc. Patrik Berger	Done
4813	Task	Oprava warningov z kompilatora Linux	Bc. Marek Rostar	Done
4815	Task	Oprava warningov cpplint	Bc. Lukas Hagara	Done
4816	Task	3d mys zmena pouzitia	Bc. Marek Rostar	Done
4820	Task	Analyza kniznic pre podporu K2 s freenect,openni	Bc. Peter Belai	Done
4822	Task	Ozivit tlacitko kinect snapshot	Bc. Peter Belai	Done
4823	Task	Opis Kinect use casov	Bc. Lukas Hagara	Done
4824	Task	Vytvorit dataset via K1	Bc. Peter Belai	Done
4840	Task	Zdokumentovat štruktúru grafu sceny	Bc. Martin Mokry	Done
4852	Task	Vytvorenie diagramu tried 1	Bc. Patrik Berger	Done
4853	Task	Vytvorenie diagramu tried 2	Bc. Martin Mokry	Done

Obrázok 8.3.2 Export úloh šprint 2 týždeň 2