Project Software-ontwikkeling en Professionele Vaardigheden

Richtlijnen analyseverslag

Universiteit Hasselt

Academiejaar 2020-2021

1. Doel

De eerste weken van het project worden besteed aan het maken van een analyse: onderzoek naar de nodige algoritmen en datastructuren met behulp van cursussen, boeken en opzoekingen op het internet. Het resultaat van deze analyse wordt beschreven in het analyseverslag. Hiernaast bevat het verslag ook een planning en een taakverdeling.

Hierdoor worden jullie gestimuleerd om, nog vóór er een regel broncode wordt geschreven, na te denken over de belangrijkste/meest complexe onderdelen van het project, waaronder datastructuren en algoritmen, UI ontwerp, een gedetailleerde planning en taakverdeling. Bovendien krijgen de begeleiders en opdrachtgever zo al vroeg een duidelijk beeld van de geplande werkwijze, zodat er, indien nodig, tijdig kan bijgestuurd worden.

2. Inhoud

Onderstaande onderdelen worden verwacht in het analyseverslag. Een verplicht te gebruiken LaTeX template voor het verslag zal beschikbaar gemaakt worden op Overleaf.

Titelpagina

Maak een duidelijk voorblad waarop minstens informatie zoals groepsnummer, namen van de teamleden, onderwerp, en academiejaar vermeld worden.

Diepgaande beschrijving van het onderwerp

Leg uit wat het einddoel is van het project. Beschrijf grondig de interpretatie van de opgave. Vermeld ook welke features van je applicatie je als prioritair beschouwt, en welke als extra's voorzien worden indien er tijd overblijft. Bespreek wat de noden van de opdrachtgever(s) zijn, en welke algemene features je project moet ondersteunen om aan die noden te voldoen. De gedetailleerde beschrijving van hoe je die features zal gaan vertalen naar een goed gedefinieerd informatica-probleem, gebeurt in de rest van dit analyseverslag. Leg alle begrippen die je gebruikt voldoende uit. Dit is de basis waarop de rest van het project verder gewerkt zal worden.

Gerelateerd werk: software, libraries en literatuur In deze sectie wordt een lijst gegeven van bestaande software die een gelijkaardige functionaliteit heeft (geheel of gedeeltelijk), van bestaande

technieken (beschreven in boeken, papers of (betrouwbare) online bronnen) om het probleem/de deelproblemen op te lossen en van implementaties (software libraries) die reeds bestaan. Geef aan wat je hieruit wel/niet meeneemt in je eigen project en waarom. Wees volledig! Kijk zowel naar functionaliteit als gebruiksvriendelijkheid, vergelijk concurrerende technieken, geef proof-of-concepts, ...

Evaluatiecriteria

In deze sectie geven jullie een prioriteitenlijst van features en criteria waaraan het eindproduct getoetst moet worden. Deze lijst moet zowel functionele als niet-functionele vereisten bevatten en zal belangrijk zijn om tijdens het schrijven van de code te controleren of je alle gevraagde functionaliteit geïmplementeerd hebt. De uitvoerigheid waarmee deze criteria zijn opgesteld zal ook invloed hebben op de eindbeoordeling van het project.

De lijst van criteria vloeit voort uit het gesprek met de opdrachtgever. Veelal zijn de aangegeven requirements van de opdrachtgever een functionele requirement, bijvoorbeeld “Een gebruiker moet een account kunnen aanmaken”. Uit deze functionele requirement ontstaat in de praktijk ook vaak een niet-functionele requirement die - al dan niet - door de opdrachtgever wordt aangehaald. Die niet-functionele requirement is inherent aanwezig door hardware constraints en/of wordt gekozen als limitatie voor het systeem om het ontwikkelingsproces te versnellen, bijvoorbeeld: “Tijdens het aanmaken van het account mag de gebruikersnaam niet groter zijn dan 255 karakters door geheugen limitaties van de database”. Functionele requirements zijn dus, met andere woorden, de flow van je applicatie en hoe deze reageert op invoer en uitvoer. De niet-functionele requirements daarentegen geven aan hoe de applicatie omgaat met deze in- en uitvoer. De zes grootste niet-functionele domeinen zijn als volgt:

- **Availability**: Requirements met betrekking tot beschikbaarheid, dit zijn veelal zeer projectspecifieke requirements. Bijvoorbeeld: De website moet 99.9% van de tijd bereikbaar zijn.
- **Interoperability**: Requirements over data-uitwisseling, bijvoorbeeld: De applicatie zal toegang geven tot interne modellen a.d.h.v. een publieke API.
- **Modifiability**: Requirements rond aanpasbaarheid van de applicatie, bijvoorbeeld: Het spel heeft een plugin systeem om nieuwe vijanden toe te voegen.
- **Performance**: Requirements met betrekking tot de responsiviteit van het programma, bijvoorbeeld: De compilatie van het gemaakte design zal plaatsvinden onder de X tijd.
- **Security**: Requirements gebonden aan veiligheid, bijvoorbeeld: Wachtwoorden moeten geëncrypteerd worden opgeslagen.
- **Testability**: Requirements die veelal gebonden zijn aan (op voorhand bepaalde) limieten, bijvoorbeeld: De website moet 1000 gebruikers tegelijkertijd aankunnen.

Analyse

In dit deel wordt de eigenlijke analyse van het project besproken. In het analyseverslag moeten zeker de volgende elementen aan bod komen:

- **Beschrijving en verantwoording van de gekozen algoritmes:** Complexe algoritmes of onderdelen ervan worden volledig toegelicht, uitgewerkt en in pseudo-code beschreven om de eigenlijke implementatie later te vergemakkelijken; alternatieven worden besproken en afgewogen.

- **Beschrijving en verantwoording van de keuze van Algoritmen/Datastructuren:** (ADT's) Wat zijn de belangrijke/complexe ADTs die in je applicatie nodig zijn? Hoe ga je ze implementeren? Indien van toepassing, verwijs ook naar low-level datastructuren die jullie wensen te gebruiken binnen jullie project (bijvoorbeeld: *std::map*). Elke keuze moet ook voldoende gemotiveerd worden; bv. waarom gebruik je een boomstructuur? Waarom is structuur X efficiënter dan structuur Y? Merk op dat de keuze van ADT's vaak gekoppeld is aan specifieke algoritmes. Je mag ervoor kiezen om de ADTs bij de algoritmen zelf toe te lichten, of hieraan een aparte sectie te wijden.
- **Ontwerp:** Geef hier de globale werking en structuur van je programma aan. Je kan ervoor kiezen om hier gebruik te maken van bestaande ontwerpstructuren zoals bijvoorbeeld klassendiagrams, usecasediagrams, objectdiagrams, etc. Belangrijk is dat hier de flow van de applicatie duidelijk (visueel) wordt weergeven opdat de onderlinge relaties van verschillende componenten duidelijk worden. Licht de belangrijke elementen uit het ontwerp ook kort toe.
- **Data en databaseschema:** (indien van toepassing) In applicaties die data opslaan, verwerken en/of (interactief) weergeven dient een inventaris gemaakt te worden van welke data er precies nodig is, wat er met die data dient te gebeuren (welke operaties dient de applicatie aan te bieden, in welke stappen wordt data verwerkt) en hoe de data intern opgeslagen gaat worden. Dit kan bijvoorbeeld gebeuren in een SQL database, of met behulp van speciale data-objecten. Vermeld deze in je verslag, en geef aan op welke manier je beslist hebt over de structuur waarin deze data zal worden bewaard.
- **Bestandsstructuur:** (indien van toepassing) Indien je applicatie gegevens van bestanden zal inlezen of naar bestanden zal wegschrijven, leg je best op voorhand al de bestandsstructuur vast. Indien er reeds bestaande formaten voor het probleem bestaan, ga dan na of het mogelijk is om dit formaat ook te gebruiken.

Algemene vereisten:

- Verantwoord bondig de belangrijkste ontwerpkeuzes aan de hand van je opgedane kennis uit de vakken Object-georiënteerd programmeren I en Object-georiënteerd programmeren II. Zorg voor een goede scheiding tussen je grafische user interface en de applicatielogica (Model-View), pas de GRASP patronen toe om verantwoordelijkheden te verdelen tussen de verschillende klassen (denk o.a. aan low coupling, high cohesion), en houd rekening met de SOLID principes (Single Responsibility, Open/Closed, Liskov substitution, Interface segregation en Dependency inversion principle).
- Beperk je op dit moment tot de meest belangrijke onderdelen van de applicatie. Beperk je bijvoorbeeld voor het GUI gedeelte enkel tot de koppeling tussen de GUI en applicatielogica, en laat de specifieke details van de GUI klassen weg. Vermeld wel zeker klassen die deel uitmaken van de eerder beschreven algoritmen.

Mockups

Geef een initieel beeld van hoe de grafische interface van de applicatie eruit zal zien met behulp van niet-interactieve paper mockups. Let erop dat de mockups concrete voorbeeldinhoud bevatten (dus geen placeholders zoals "lorem ipsum" tekst of lege afbeeldingen). Het is voldoende om de gemaakte schetsen als ingescande afbeeldingen toe te voegen aan het analyseverslag zolang ze leesbaar zijn.

Taakverdeling en planning

Op basis van de analyse is het duidelijk welke onderdelen ontworpen moeten worden. Het is belangrijk om de taken op voorhand te verdelen en hier ook een planning aan te koppelen.

Stel een aantal mijlpalen voorop die stap voor stap tot een afgewerkte oplossing leiden. Stel hierbij de juiste prioriteiten! Welke vereisten vormen de kern van de applicatie? Welke afhankelijkheden zijn er? Zorg ervoor dat iedereen een evenwichtig takenpakket heeft gedurende de implementatie van het project. Een taakverdeling waarbij één persoon de volledige GUI implementeert en een andere persoon de overige code is niet toegelaten. Koppel ook deadlines aan mijlpalen en geef elk groepslid voor elke mijlpaal een aantal duidelijk afgelijnde taken.

Maak de planning en taakverdeling zo gedetailleerd mogelijk! Het is onvoldoende te vermelden dat "*teamlid A tijdens het eerste trimester werkt aan de zoekboom*". Schrijf dat "*teamlid A van dag X1 tot dag Y1 werkt aan het invoegen van nodes, van X2 tot Y2 aan het testen van het verwijderen van nodes, ...*".

Let op: zorg ervoor dat de planning realistisch is; laat voldoende ruimte voor onderdelen die later bijkomend onderzoek kunnen vereisen, en plan expliciet voldoende tijd om grondig te testen.

Bibliografie

Een lijst van geraadpleegde werken. Zorg voor duidelijke referenties in de tekst, niet enkel een lijst aan het einde.

3. Belangrijke opmerking ter afsluiting

Hou er rekening mee dat het analyseverslag voor een aanzienlijk deel van de punten meetelt en dus ook essentieel is voor een goed eindresultaat. D.w.z. dat een vaag beschreven analyseverslag ook invloed heeft op de resterende elementen (zoals b.v. de broncode) later in het project.

Omwille van het belang van deze analysefase, wordt de feedback op het analyseverslag verplicht verwerkt in een revisie van het verslag, voordat er wordt gestart met de eigenlijke implementatie.