Robots Kinematica 2324 S2 peerreview-instructie

Joost Kraaijeveld en Chris van Uffelen

21 november 2024

1 Inleiding

Dit is de peerreviewopdracht voor WoR-Robots. In een tijdsbestek van 2 uur moet je een oordeel formuleren over de kwaliteit, bruikbaarheid en onderhoudbaarheid van de codebijdrage van een van je medestudenten. Lees deze instructie ruim van te voren een keer door om er voor te zorgen dat je goed voorbereid bent. Het is jouw verantwoordelijkheid om ervoor te zorgen dat je goed voorbereid bent op de review. Installeer in ieder geval cppcheck en/of clang-tidy om de kwaliteit van de code te kunnen beoordelen.

2 Review-materiaal

Je krijgt het te reviewen materiaal in de vorm van een tar.xz-bestand. Een dergelijk archief is waarschijnlijk uit te pakken met behulp van de file-manager van je OS maar als het daarmee niet lukt dan kan het altijd in een terminal met behulp van tar. In het bestand zit de broncode die je moet beoordelen. Eventueel zit er nog een readme.txt bij met opmerkingen of wijzigingen. Doordat je beide versies hebt is het tamelijk triviaal om met de meeste IDE's een vergelijking te maken tussen de beide versies zodat je de eventuele wijzingen goed kunt bekijken.

3 Review-tools

Je bent vrij om iedere tool te gebruiken die je zelf nodig acht om een goede review uit te kunnen voeren. Je moet in je review melding maken van de gebruikte tools. Van ieder tool die je gebruikt moet de uitvoer als bijlage worden meegeleverd. Dat geldt sowieso voor de complete output van de compilatie. Ook moet tenminste 1 statische codechecker gebruikt worden. Voor de hand liggende keuzes hiervoor zijn cppcheck of clang-tidy. Voor die laatste keuze is een compilatie-database noodzakelijk: die moet je zelf maken.

4 Essentie van de review

De review gaat alleen over de code die je medestudent heeft geschreven. De oorspronkelijke robotwereldopdrachtcode hoef je dus niet te reviewen.

De essentie van een peerreview is een uitpraak te doen over de kwaliteit, bruikbaarheid en onderhoudbaarheid van de code. Het helpt de auteur niet wanneer je review alleen maar bestaan uit: “ik vind dit niet goed” en “ik zou dat anders doen”.

Onderbouw beweringen en geef aan hoe iets beter kan. Dit kan door in essentie de volgende reeks te hanteren:

- Waarneming: geeft concreet aan waar je je op baseert.
- Conclusie: geef aan welke conclusie je trekt uit de waarnemingen en waarom je tot die conclusie komt. (Een conclusie is niet alleen een samenvatting van de waarnemingen maar voegt ook een onderbouwde waardeoordeel aan, bijvoorbeeld: “de inconsequente codestijl verlaagt de bruikbaarheid van de code omdat de code veel minder gemakkelijk te lezen is.”).
- Aanbeveling: geef aan hoe de auteur zijn code kan verbeteren.

5 De review-onderdelen

Hier volgen een aantal inhoudelijke aanwijzingen die gelden voor de review. Mocht je bij het reviewen dingen tegen komen die niet in de aanwijzingen staan maar die jij wel relevant vindt dan moet je die uiteraard ook opnemen.

5.1 Functionaliteit en Requirements

Controleer of de code de requirements implementeert. Het is een pre wanneer de auteur hier zelf de garanties voor geeft in de vorm van tests.

Een voorwaarde is dat de auteur het de reviewer zo gemakkelijk mogelijk maakt de applicatie te kunnen bouwen en uitvoeren.

5.2 Kwaliteit en onderhoudbaarheid

In dit stuk staan een aantal expliciete (maar niet uitputtende) aandachtspunten die gebruikt kunnen worden bij het houden van een (peer-)review op het gebied van code. Deze zijn aanvullend op eventuele andere punten die in bijvoorbeeld de JSF-styleguide, CppCoreGuidelines of andere style-guides zijn opgenomen. Als er verschil van mening is tussen dit stuk en de diverse style-guides dan is de CppCoreGuidelines veelal leidend.

5.2.1 Applicatie

- Duidelijke compilatieinstructies, e.g.:
  - ../configure ; make.
  - cmake . ; make.
  - een volledig compilatiescript.
- Volledige output van de compilatie, inclusief gebruikte commandlines, is te zien (CMake: make VERBOSE=1 of cmake -DCMAKE_VERBOSE_MAKEFILE:BOOL=ON).
- Geen compiler warnings of errors op het hoogste warning niveau (e.g. -Wall -Wextra -Wconversion).
- CPPCheck is via make aan te roepen (e.g. make cppcheck).
- Volledige output van cppcheck, inclusief gebruikte commandlines, is te zien.
- Geen ccpcheck warnings of errors op het hoogste warning niveau (e.g. –enable=all –inconclusive voor zowel de h/hpp als cpp files).
- Duidelijke instructies voor het opstarten van het programma (directory, commando, argumenten etc).

5.2.2 Algemene code

- Consistente code-stijl.
- Orde en netheidheid (consistentie in de layout, variabele namen, indentie, lijnlengte, commentaar etc).
- Geen uitgecommentaarde restant-code.
- Geen overbodige includes in headers of sources (e.g. controle door Eclipse via het menu Source -> organize includes, shift+ctrl+o, leidt niet tot wijzigen includes).
- Gebruik forward declarations waar mogelijk.

5.2.3 Namespaces

- Gebruik van logische namespaces (naamgeving, vulling met classes en interfaces en Doxygen).
- High cohesion in de namespace, low coupling tussen namespaces (e.g. blijkt uit bijvoorbeeld uit een UML-diagram).
- Coupling via interfaces (volledig abstracte classes) en niet via niet-abstracte classes.

5.2.4 Klassen

- Single responsibility van klassen, blijkend uit de specificatie, de naamgeving en Doxygen-commentaar.
- Iedere klasse heeft zijn eigen header en source.
- Een eventuele uitzondering hierop zijn kleine klassen die *alleen* nodig zijn voor een grote klasse die gedeclareerd en geïmplementeerd kunnen worden bij de grote klasse.
- Een klasse is altijd onderdeel van een namespace.

5.2.5 Functies

- Geen lange functies (e.g. meer dan 1 scherm is te groot, i.e. minder dan 40/50 regels, zie hier voor tooling).
- Geen complexe functies (e.g. cyclomatic complexity <=10 of <=15, 10 heeft voorkeur, zie hier voor tooling).
- Single responsibility van functies, blijkend uit specificatie (pre- en postconditie), naamgeving en Doxygen-commentaar.
- Pre- en postcondities van functies worden gecheckt (in ieder geval in debug mode).
- Argumenten worden alleen als pointer doorgegeven als argument aan externe libraries of functies die een pointer verwachten.
- Geen (onnodige) pass by value voor UDCs (let op, cppcheck faalt in een aantal gevallen en je zult dus ook met de hand moeten kijken).
- Shared_ptr's worden bij value gepassed.
- Gebruik typesafe interfaces, e.g. gebruik een class IPAddress class en geen std::string als IP-address in vermomming (anders dan waarschijnlijk in de ctor van IPAddress).
- Alle publieke functies moeten buiten de class gebruikt worden, i.e. geen overbodige publieke functies.

5.2.6 Variabelen

- Gebruik van zinvolle variabele namen.
- Gebruik van RAII.

6 Inleveren

Aan het eind verstuur je een mail naar de docent met daarin je review in een *pdf*-bestand en alle output van de gebruikte tools en compilatie als bijlage. De bijlages mogen ascii-bestanden zijn of, als het schermafbeeldingen zijn, png dan wel vergelijkbare bestanden. Ook moet je een zip (of tar.gz of zo) met alleen de *src-directory* van de door jou gereviewde code *zonder de gegenereerde bestanden* als bijlage toevoegen. Vervolgens upload je het zelfde via ISAS.