Het bepalen van de onderhoudbaarheid van objectgeoriënteerde broncode door middel van metrieken

Publicatiestatus: *Openbaar*

Datum: *10 september 2007*

Student: *C.B. Woolderink*

Opleiding: *Master Software Engineering*

Instelling: *Universiteit van Amsterdam*

Bedrijf: *Info Support b.v.*

Afstudeerdocent: *Prof. Dr. J. van Eijck*

Technisch begeleider: *Drs. M. Pil*

Opdrachtgever (1): *Ing. B. Meerman*

Opdrachtgever (2): *Ing. D. Joosten*

Samenvatting

Binnen Info Support is de behoefte om een enkele meetwaarde te hebben voor softwarekwaliteit, die op basis van automatische codeanalyse verkregen kan worden. Om de scope echter beperkt te houden is ervoor gekozen om enkel te focussen op het bepalen of de onderhoudbaarheid van objectgeoriënteerde broncode door middel van codemetrieken gemeten kan worden.

Het begrip onderhoudbaarheid kent verschillende definities en kan vanuit verschillende invalshoeken bekeken worden. Om hierover duidelijkheid te scheppen is een nieuwe definitie voorgesteld.

Er zijn meerdere onderzoeken uitgevoerd om de relatie tussen metrieken en onderhoudbaarheid aan te tonen, echter wel vanuit verschillende definities dan in deze scriptie. Na het analyseren van verschillende metrieken is gebleken dat een verhoging van de meetwaarde zowel positief als negatief uitgelegd kan worden voor onderhoudbaarheid. Dit komt omdat de semantiek van de code niet terugkomt in de meetwaarde. Bestaande onderzoeken lijken soms te zijn “gebiassed”, omdat dit aspect buiten beschouwing blijft.

Om tot een enkele meetwaarde voor onderhoudbaarheid te komen, is gekeken naar de twee soortgelijke benaderingen Static Confidence Factor en Maintainability Index. Bij beiden wordt een kwaliteitsattribuut vertaald naar een enkele waarde. Het is echter niet duidelijk waarin deze meetwaarden uitgedrukt kon worden, zodat het kwaliteitsattribuut zelf dus niet gemeten wordt.

Een experiment in een laboratorium bood de mogelijkheid om zo dicht mogelijk bij de nieuwe definitie van onderhoudbaarheid te blijven. De bestede tijd kan namelijk gelijk getrokken worden aan onderhoudbaarheid, omdat overige factoren constant blijven in de gecontroleerde omgeving.

Het experiment bestaat uit drie kleinere experimenten, die elk een voorbeeld, tegenvoorbeeld en een ijking probeert te bewerkstelligen. Het voorbeeld, dat de metriek zoals verwacht een relatie heeft tot onderhoudbaarheid, kwam niet naar voren uit de resultaten. Er was namelijk geen verschil te onderkennen tussen twee verschillende versies. Het tegenvoorbeeld kon niet worden aangetoond, omdat het experiment uitschieters bevatte, die niet gerelateerd waren aan de code. De ijking gaf bij de replicatie dezelfde trend weer als het oorspronkelijke experiment, echter wel met een groot verschil in bestede tijd.

Het experiment heeft een signaal afgegeven dat een verschuiving van de metrieken geen invloed hoeft te hebben op onderhoudbaarheid en dat daarom voorzichtig moet worden omgegaan met codemetrieken. Metrieken confronteren je met feiten over de code, zodat erover nagedacht gaat worden. Het zijn dus signalen, maar moeten dus wel gecontroleerd worden door een expert.

In de reflectie van het experiment zijn de interne, externe en “construct”-validiteit geanalyseerd. Hieruit blijkt dat de resultaten van het onderzoek voorzichtig geïnterpreteerd moeten worden. Interessant toekomstig onderzoek is het op grotere schaal uitvoeren van een soortgelijk experiment, waarbij de uitgevoerde aanpak hergebruikt kan worden.

Voorwoord

Achtergrondinformatie

Deze scriptie beschrijft het Masterproject (24 ects), dat uitgevoerd is door Chris Woolderink voor de eenjarige masteropleiding Software Engineering aan de Universiteit van Amsterdam. Het onderzoek is uitgevoerd in opdracht van Info Support b.v. te Veenendaal.

In de maanden januari, februari en maart is hieraan deeltijd gewerkt. Van april t/m begin september is een voltijdse inspanning geleverd.

Dankwoord

Ten eerste wil ik mijn familie en vrienden bedanken voor hun steun. Ik bedoel hiermee met name het begrip dat ik te weinig tijd met hun heb kunnen doorbrengen gedurende de gehele masteropleiding.

Vervolgens wil ik het complete docententeam van de masteropleiding Software Engineering bedanken, met in het bijzonder Jan van Eijck en Jurgen Vinju voor de begeleiding tijdens afstudeertraject. De goede inzet, nauwe betrokkenheid en onderlinge samenwerking gedurende het hele jaar heeft een positief effect gehad op deze scriptie.

In de beginperiode van het afstudeertraject heb ik contact gehad met Chris Verhoef, die als hoogleraar is aangesloten aan de Vrije Universiteit Amsterdam. Hij heeft me goed duidelijk gemaakt dat onderhoudbaarheid lastig te definiëren is. Hierdoor ben ik extra kritisch gaan kijken naar onderhoudbaarheid, waardoor er een nuttig “kapstokje” ontstond voor het vervolg van het onderzoek .

Vanuit Info Support wil ik in het bijzonder Bas Meerman bedanken, vanwege de heldere communicatie en de grote hoeveelheid tijd die hij beschikbaar heeft gesteld. Ook Dennis Joosten, die op de achtergrond toekeek en indien nodig Bas Meerman verving tijdens zijn afwezigheid, heeft een belangrijke rol gespeeld.

De inhoudelijke gesprekken met Marco Pil hadden altijd een positieve uitwerking. Ook de voortgangsbesprekingen met Pascalle Hijl zorgde ervoor dat alles goed op de rails bleef. Ik wil hun daarvoor dan ook expliciet bedanken.

Daarnaast wil ik het bedrijf Info Support met klem bedanken voor de mogelijkheid van het uitvoeren van dit afstudeeronderzoek. Bovendien is het beschikbaar stellen van een cursuslokaal voor het experiment gedurende twee dagen (voorbereiding en uitvoering) erg waardevol geweest voor het onderzoek.

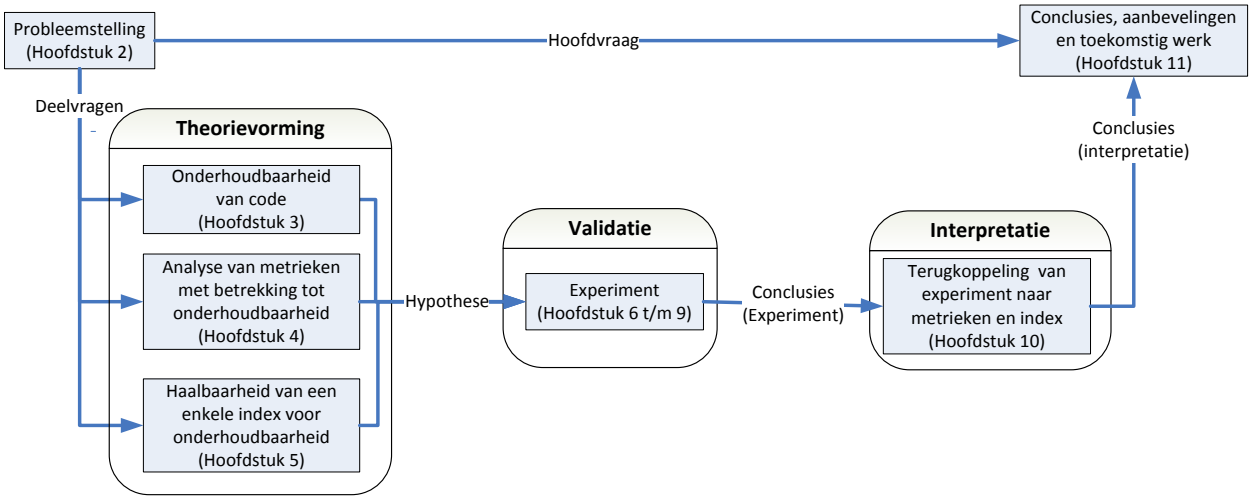
Tot slot wil ik de deelnemers van het experiment bedanken, te weten: Alex van Beek, Chris van Beek, Aldert Boerhoop, Wim van Gool, Danny van Heumen, Niels Naglé, Mark Niemantsverdriet, Bastiaan Pierhagen, Maarten Schilt en Ronnie de Vaan.

Chris Woolderink

Veenendaal, september 2007

Inhoudsopgave

1. Inleiding



Figuur 1 Schematische weergave van de samenhang tussen de hoofdstukken

In Figuur 1 is schematisch de structuur van de hoofdstukken weergegeven.

Hoofdstuk 2 bespreekt de probleemstelling. Er wordt een context geschetst, waarin bepaalde vraagstukken liggen. Vervolgens wordt de scope afgebakend, zodat een hoofdvraag met onderliggende deelvragen geformuleerd worden, waarop in de scriptie verder wordt ingegaan.

Elke deelvraag wordt apart beantwoord in de daarop volgende hoofdstukken, zodat er een theoretische basis wordt gelegd. In Hoofdstuk 3 wordt een eigen definitie gegeven van onderhoudbaarheid, die gebruikt wordt in de rest van de scriptie. In Hoofdstuk 4 worden bestaande metrieken uitgelegd en geanalyseerd. De haalbaarheid van een enkele meetwaarde voor onderhoudbaarheid komt in Hoofdstuk 5 aan bod.

De theorie wordt vervolgens gevalideerd door middel van een experiment. De opzet, uitvoering, resultaten, conclusies en reflectie staan beschreven in de hoofdstukken 6 t/m 9.

In Hoofdstuk 10 wordt afstand genomen van het experiment, zodat bepaald kan worden wat de betekenis is voor metrieken in het algemeen en de enkele meetwaarde. Hier vindt dus een terugkoppeling plaats naar de theorievorming, door de resultaten van het experiment te interpreteren in een bredere context.

Tot slot wordt in Hoofdstuk 11 de eindconclusies getrokken. Ook worden aanbevelingen gedaan over praktische vervolgstappen voor Info Support. Bovendien wordt mogelijk toekomstig werk voorgesteld aan de wetenschap.

In de Bijlagen A t/m P staan aanvullend materiaal, die gebruikt is voor het experiment. Dit schept openheid over het experiment, zodat deze op waarde ingeschat kan worden.

2. Probleemstelling

In dit hoofdstuk wordt de achtergrond en context geschetst, waarin bepaalde vraagstukken liggen. Vervolgens wordt de scope afgebakend, zodat een hoofdvraag met onderliggende deelvragen geformuleerd worden, waarop de scriptie antwoord gaat geven.

Achtergrond

Info Support

Info Support is een IT-dienstverlener met ruim 230 medewerkers in Nederland. Het dienstenpakket omvat Training, Professional Development, Life Cycle Management en Hosting. De markten waarin geopereerd worden zijn Handel en Industrie, Finance, Overheid en Zorg en Verzekeringen. De focus ligt op de technologieën Java en .NET.

Het onderzoek is uitgevoerd in opdracht van het Professional Development Center (PDC), dat zich richt op het opzetten van een professionele werkwijze binnen Info Support. Dit heeft geresulteerd in de softwareontwikkelstraat Endeavour.

Endeavour

Endeavour is een omgeving die de complete levenscyclus van een product ondersteunt, in het bijzonder ontwikkeling, onderhoud en hergebruik, en daarbij gebruik maakt van een productieproces en productiemiddelen die beproefd, gestandaardiseerd en gestructureerd zijn. De focus ligt op administratieve applicaties.

Endeavour bevat een verzameling van producten, welke staan weergegeven in Tabel 1.

Categorie	Toelichting
Tools	Specifieke tools ter ondersteuning van het softwareontwikkelproces.
Building Blocks	Componenten ontwikkeld met het oog op hergebruik. Denk bijvoorbeeld aan data access, logging of exception management.
Standaards en richtlijnen	Afspraken betrekking op programmacode en documenten. Handleiding voor installeren van ontwikkelmachines en hoe de infrastructuur voor de ontwikkelstraat in te richten.
Referentiearchitectuur	Opgedeeld in een logische en technische referentiearchitectuur, zodat concepten en specifieke technologieën losgekoppeld zijn.
Processen	Hierin staat de ontwikkelcyclus vastgelegd.
Trainingen	Trainingen voor ontwikkelaars, architecten en projectmanagers.

Tabel 1 De verschillende categorieën waarin de producten van Endeavour .NET zijn opgedeeld

Context

De opgedane ervaring met Endeavour stelt Info Support in staat om een ontwikkelstraat aan te bieden die garanties biedt met betrekking tot de snelheid (uren per functiepunt) en de kwaliteit (fouten per functiepunt) van de ontwikkelde software. Binnen Endeavour wordt gebruik gemaakt van standaarden en diverse applicaties om de kwaliteit van de software te waarborgen.

Kwaliteit is echter meer dan enkel het aantal fouten per functiepunt. Ook het garanderen dat code aan voorafgestelde standaarden voldoet betekent niet per definitie dat het kwalitatief in orde is. Info Support heeft behoefte om een completer beeld te krijgen over de softwarekwaliteit en deze dagelijks en op automatische wijze toegankelijk te maken voor managers en projectleiders.

Ook wordt er door externe bedrijven een beroep gedaan op Info Support om code in beheer te nemen. Info Support heeft dus dagelijks te maken met grote hoeveelheden broncode. Om hiervan de kwaliteit te bepalen moet er handmatig door een expert naar gekeken worden. De belangrijkste nadelen hiervan zijn de benodigde tijd en kosten.

De wens is om deze nadelen (deels) weg te nemen door middel van automatische codeanalyse, zodat er goedkoop en snel een enkele kwaliteitsbeoordeling over de broncode gegeven kan worden. Ook kan dit geïntegreerd worden in het dagelijkse buildproces van Endeavour, zodat hierover dus dagelijks gerapporteerd kan worden.

De vraagstelling die hieruit kan worden afgeleid is:

Hoe kan de kwaliteit van broncode middels automatische codeanalyse gemeten worden?.

Deze vraag is echter te breed geformuleerd om binnen de gestelde tijd van dit onderzoek te beantwoorden. Om deze reden wordt de scope verder afgebakend op de onderstreepte begrippen.

Kwaliteit

Kwaliteit is een breed begrip. Om een compleet beeld te vormen worden bestaande kwaliteitsmodellen toegelicht met daarbij de focus op codekwaliteit.

Garvin

De term kwaliteit heeft verschillende betekenissen voor verschillende mensen. Het valt te vergelijken met bijvoorbeeld schoonheid, dat per persoon verschilt. David Garvin heeft onderzocht hoe kwaliteit gezien wordt in verschillende domeinen, zoals filosofie, economie en marketing (Garvin, 1988). Hij kwam tot de conclusie dat kwaliteit een complex concept is met meerdere facetten. In Tabel 2 staan vijf verschillende benaderingen die hij onderkend heeft.

Benadering	Toelichting
Transcendent	Het is niet duidelijk wat kwaliteit is, maar het is iets goeds. Het is wel te ervaren en zorgt voor een hoge mate van succes.
Productgericht	Het product heeft iets dat gelijkwaardige producten niet of minder hebben. Hierbij wordt gekeken naar productspecifieke eigenschappen, die meetbaar zijn.
Gebruikersgericht	De geschiktheid van het product voor gebruik en of het overeenkomt met de verwachtingen van de gebruiker.
Productiegericht	Of het product conform het ontwerp, specificaties of eisen is.
Waardegericht	Het product is de beste combinatie van prijs en functies.

Tabel 2 Vijf verschillende definities van kwaliteit volgens Garvin

Een transcendente benadering is niet gewenst, omdat kwaliteit dan niet gedefinieerd kan worden. Het is juist de bedoeling dat er een duidelijk beeld ontstaat van de codekwaliteit voor managers en projectleiders.

Bij de productgerichte benadering worden eigenschappen gemeten van een product. Denk bijvoorbeeld aan de hoeveelheid vlees die in een bepaalde kroket zit, welke vergeleken kan worden met andere kroketten. Hoe meer vlees verwerkt is, des te beter de kwaliteit is. Op eenzelfde manier kunnen eigenschappen van code gebruikt worden om een uitspraak over de kwaliteit te geven. Een voordeel is dat deze eigenschappen middels automatische analyse verkregen kunnen worden.

De gebruikersgerichte benadering zegt iets over de geschiktheid van gebruik en of het voldoet aan de verwachtingen van de gebruiker. De doelgroep van het “product” broncode is in deze scriptie de onderhoudsprogrammeur. Het gaat dus niet om een normale eindgebruiker.

Aangezien software volgens Laws of Software Evolution continu verandert en groeit (Lehman, et al., 2006), moeten er telkens aanpassingen worden doorgevoerd door een (onderhouds)programmeur. In feite wordt hier de onderhoudbaarheid van broncode bedoeld, waar later in deze scriptie dieper op in wordt gegaan. Bij het schrijven van code moeten programmeurs als doelgroep beschouwd worden, vanwege het onderhoud dat hierop later plaatsvindt (McConnell, 2004).

De normale eindgebruiker van een programma wordt dus voor dit onderzoek niet meegenomen in de gebruikersgerichte benadering, want de focus ligt enkel op codekwaliteit. Een programma kan namelijk niet aan de verwachtingen voldoen van een eindgebruiker, terwijl de codekwaliteit goed is. Maar andersom ook, dat een eindgebruiker tevreden is over het programma, maar dat de codekwaliteit beneden peil ligt.

De productiegerichte benadering bekijkt of het product conform het ontwerp of eisen ontwikkeld is. Documentatie over het ontwerp of eisen moeten beschikbaar en terug te koppelen zijn aan code. Enkel code als input is dus niet voldoende, terwijl de scope van het onderzoek ligt op enkel automatische codeanalyse. Om deze reden wordt deze benadering verder buiten beschouwing gelaten.

Ook bij de waardegerichte benadering is meer benodigd dan code, want de gemaakte ontwikkelkosten moeten beschikbaar zijn. Hoewel software zeker een prijs heeft, blijft het financiële aspect verder buiten beschouwing in deze scriptie.

Voor het “product” code zijn productgerichte en gebruikersgerichte benadering van Garvin interessant voor dit onderzoek en hierop wordt de scope dan ook afgebakend.

Benadering	Toelichting
Productgericht	Door het selecteren van codemetrieken kan code op basis van de meetwaarden vergeleken worden.
Gebruikersgericht	De (onderhouds)programmeur als gebruiker van de code, omdat deze tijdens de levensduur van de applicatie aanpassingen moet doorvoeren.

Tabel 3 Relevante benaderingen volgens Garvin voor codekwaliteit

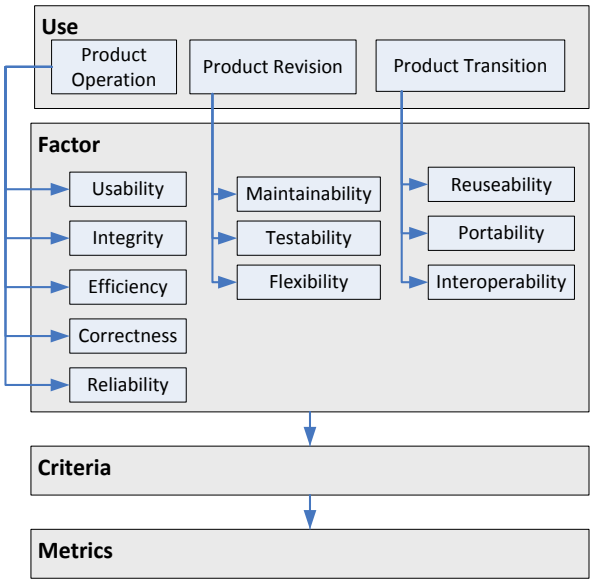
Softwarekwaliteit

Naast een algemene kwaliteitsmodel zoals van Garvin, zijn er ook specifieke kwaliteitsmodellen voor software opgesteld.

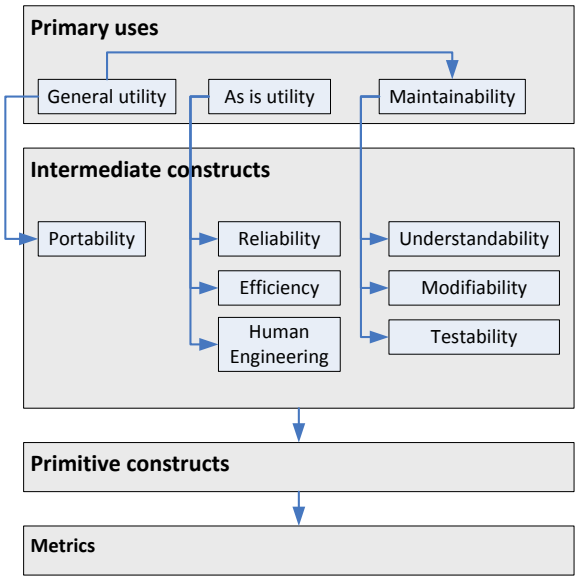
Intern attribuut zegt over het product zelf, terwijl extern attribuut iets zegt over de uitwerking van het product. Veel van deze modellen maken hiertussen geen onderscheid. Daardoor is lastiger te begrijpen wat precies onder codekwaliteit verstaan wordt, dat in een intern attribuut is.

De modellen zijn wel nuttig voor het benadrukken wat belangrijk is en welke overeenkomsten er zijn.

McCall en Boehm



Figuur 2 Kwaliteitsmodel McCall



Figuur 3 Kwaliteitsmodel Boehm

Voorbeelden van twee geaccepteerde modellen zijn die van McCall (Cavano, et al., 1978) en Boehm (Boehm, et al., 1978). Deze zijn respectievelijk weergegeven in Figuur 2 en Figuur 3. Beide modellen beschrijven kwaliteit door middel van een decompositie in eigenschappen en relaties op vier gelijkwaardige niveaus.

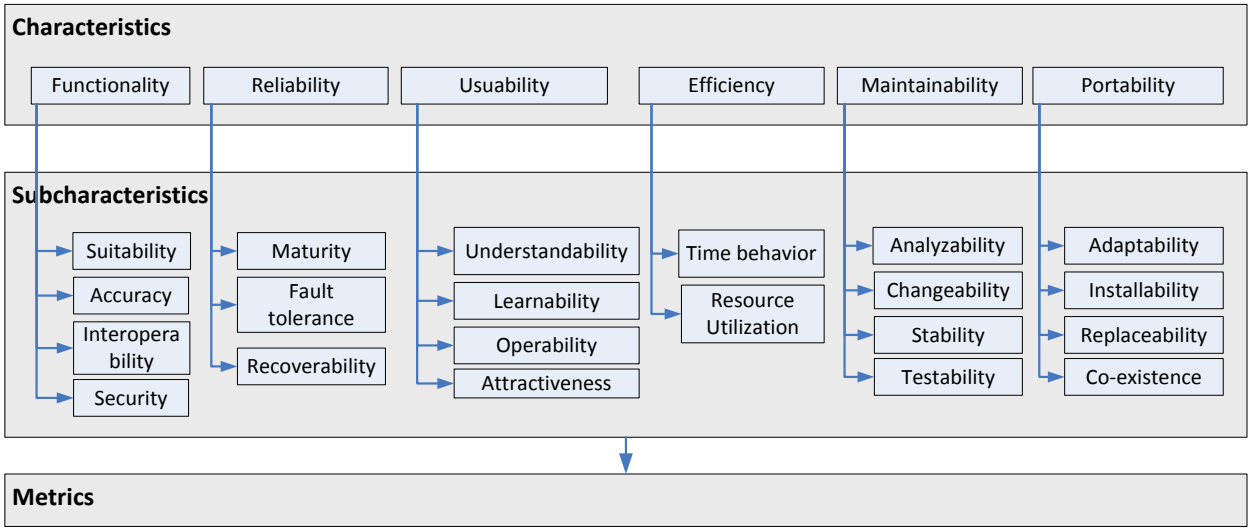
In de modellen wordt op het hoogste niveau gesproken over “Use” en “Primary uses”. De onduidelijkheid is wie de doelgroep van dit gebruik is, want dit kan de eindgebruiker van een uitvoerbaar programma en/of de (onderhouds)programmeur van de code zijn. Zoals eerder vermeld wordt in deze scriptie enkel de (onderhouds)programmeur als doelgroep beschouwd.

In beide modellen wordt broncode en uitvoerbare programmacode door elkaar gebruikt. Bij McCall gaat “Product Operation” en “Product Transition” om uitvoerbare code, terwijl “Product Revision” weer iets zegt over broncode. Bij Boehm is iets soortgelijks waar te nemen, waarbij “General Utility” en “As is utility” het gebruik van het programma door een eindgebruiker betreft en “maintainability” alleen over broncode gaat.

Verder is te zien dat in het model van Boehm testbaarheid (testability) een onderdeel is van onderhoudbaarheid (maintainability), terwijl bij McCall testbaarheid op hetzelfde niveau ligt als onderhoudbaarheid. Hieruit blijkt al dat verschillend tegen onderhoudbaarheid wordt aangekeken.

ISO 9126

Een recenter model voor de kwaliteit van een softwareproduct is ISO/IEC 9126 (ISO/IEC JTC 1, 2001). Hierin zijn de elf factoren uit (Cavano, et al., 1978) gereduceerd tot zes eigenschappen, welke hieronder zijn weergegeven.



Figuur 4 Kwaliteitsmodel ISO 9126

Ook in dit model is er nog steeds een onduidelijke scheiding tussen de broncode en uitvoerbare code. Zo vertelt “Understandability” (onder “Useability) hoe gemakkelijk een eindgebruiker de interface van het programma kan bedienen. “Analyzability” (onder “Maintainability”) is bijvoorbeeld het gemak waarmee de broncode geanalyseerd kan worden door de programmeur.

Onderhoudbaarheid (maintainability) is het enige aspect dat over de broncode zelf gaat. De andere aspecten gaan over de uitvoerbare code, dus hoe het zich gedraagt in een bepaalde situatie en omgeving.

Uitvoerbare code is natuurlijk een afgeleide van broncode, zodat specifieke keuzes in de broncode directe invloed kunnen hebben op het gedrag van het systeem. Denk bijvoorbeeld aan het selecteren van een algoritme, dat uiteindelijk invloed kan hebben op de efficiëntie (efficiency) van het uitvoerbare programma.

Om de scope echter beperkt te houden, wordt kwaliteit afgebakend op het aspect onderhoudbaarheid.

Broncode

Info Support richt zowel op Java als C#. Dit zijn beide objectgeoriënteerde talen en kennen dus dezelfde concepten zoals inheritance, polymorfisme en encapsulatie.

De scope van het onderzoek ligt op objectgeoriënteerde broncode.

Automatische analyse

Statische en dynamische codeanalyse

Code kan automatisch geanalyseerd worden door middel van statische en dynamische codeanalyse. Bij statische analyse worden feiten uit de code ontrokken zonder deze uit te voeren, terwijl dit bij dynamische codeanalyse wel het geval is.

Beide technieken bieden interessante mogelijkheden om informatie te verschaffen over de bovenstaande kwaliteitsattributen in Figuur 4.

Dynamische codeanalyse kan inzicht geven in het gedrag van het programma, dus het uitvoerbare programma. Denk bijvoorbeeld aan het monitoren van de efficiëntie of betrouwbaarheid tijdens de executie van het programma. Ook kan gedacht worden aan het genereren van rapporten over de “code coverage”, zodat bijvoorbeeld inzichtelijk kan worden gemaakt welk percentage van de code geraakt wordt door het uitvoeren van de testcode.

Statische codeanalyse wordt uitgevoerd op niet-uitvoerbare code, dus de broncode zelf.

De combinatie van statische en dynamische codeanalyse kan gebruikt worden om dode code te extraheren, zodat deze niet meer onderhouden hoeft te worden (Van Willegen, 2006).

Aangezien de focus van dit onderzoek op de broncode ligt en vanwege de beperkte tijd, wordt de scope afgebakend op statische codeanalyse.

Codemetrieken

Op basis van statische codeanalyse kan interessante informatie gevonden worden over onderhoudbaarheid.

Een voorbeeld hiervan is het automatisch detecteren van Design Patterns (Gamma, et al., 1995). Shi et al. heeft hiervan een implementatie gemaakt voor Java-broncode (Shi, et al., 2006).

Design Patterns zijn specifieke patronen, die oplossingen bieden voor veelvoorkomende problemen in objectgeoriënteerde ontwerpen. Deze standaard oplossingen zijn veel gebruikt. Door het geven van specifieke namen ontstaat er een gemeenschappelijke vocabulaire onder ontwikkelaars, zodat op een abstracter niveau de broncode geanalyseerd kan worden.

Een ander voorbeeld is het detecteren van zogenaamde “bad code smells”, die kunnen duiden op minder onderhoudbare code (Fowler, 1999). Deze worden ook wel anti-patterns genoemd, omdat het juist ongewenste patronen zijn.

Codemetrieken lijken steeds meer in populariteit te winnen. Dit blijkt bijvoorbeeld uit de keuze van Microsoft om codemetrieken te integreren in de toekomstige versie van Visual Studio. In tegenstelling tot Design Patterns en antipatterns, leveren codemetrieken meetwaarden op, die telkens in hoogte verschillen. Het is echter onduidelijk wat specifieke meetwaarden betekenen en waar grenswaarden liggen. Binnen Info Support is er een behoefte aan een enkele meetwaarde voor softwarekwaliteit; voor dit onderzoek wordt zich echter beperkt tot onderhoudbaarheid. Een interessante vraag is of deze afzonderlijke meetwaarden geïntegreerd kunnen worden in een enkele meetwaarde.

Om de scope beperkt te houden is ervoor gekozen om enkel te richten op codemetrieken.

Onderzoeksvragen

Op basis van de afgebakende scope kan de volgende hoofdvraag voor dit onderzoek geformuleerd worden:

Kan de onderhoudbaarheid van objectgeoriënteerde broncode middels codemetrieken gemeten worden?

Deze hoofdvraag is opgesplitst in de volgende deelvragen:

1. Wat wordt precies onder onderhoudbaarheid van code verstaan?
2. Welke metrieken met betrekking tot onderhoudbaarheid kunnen op basis van statische analyse van objectgeoriënteerde code verkregen worden?
    a. Wat betekenen specifieke meetwaarden van deze metrieken, waar liggen de grenswaarden?
3. Kunnen de meetwaarden van verschillende metrieken worden geconsolideerd tot een enkele meetwaarde voor onderhoudbaarheid?

3. Onderhoudbaarheid van code

Dit hoofdstuk behandelt de eerste deelvraag uit de probleemstelling welke als volgt luidt:

Wat wordt precies onder onderhoudbaarheid van code verstaan?

Ten eerste wordt gekeken naar softwareonderhoud in het algemeen en op basis hiervan wordt toegewerkt naar een definitie van onderhoudbaarheid.

Softwareonderhoud

Softwareonderhoud is het proces van het aanpassen van een softwaresysteem of component na oplevering om:

- **fouten te corrigeren,**
- **performance (of andere attributen) te verbeteren**
- **aan te sluiten op een veranderde omgeving. (IEEE, 1993)**

Definitie van softwareonderhoud volgens IEEE

Uit de definitie van IEEE blijkt dat softwareonderhoud een activiteit is die pas na oplevering plaatsvindt (IEEE, 1993). Het start pas als het systeem opgeleverd is aan de klant en omvat alle activiteiten om het systeem operationeel te houden en aan de behoeften van de klant te blijven voldoen. Dit beeld heeft relatie tot het klassieke watervalmodel, waarbij operatie en onderhoud als laatste stap is opgenomen.

Sommige auteurs zijn het niet eens dat softwareonderhoud pas begint wanneer een systeem operationeel wordt gesteld. Zo stelt Pigoski dat het onderhoud begint wanneer het ontwikkelen van start gaat.

Softwareonderhoud is het totaal van activiteiten die benodigd zijn om kosteneffectieve ondersteuning te bieden voor een softwaresysteem. Activiteiten worden zowel voor als na de oplevering uitgevoerd. Activiteiten voor oplevering betreffen het plannen van activiteiten die na oplevering plaatsvinden, mate van ondersteuning en vaststelling van logistiek. Activiteiten na oplevering betreffen aanpassen van software, geven van training en beschikbaar stellen van een helpdesk.

Definitie van softwareonderhoud volgens Pigoski

De onderzoeksvraag zegt iets over onderhoudbaarheid van code, dus het hoeft nog niet over een compleet opgeleverd softwareprogramma te gaan, die te gebruiken is door een eindgebruiker. De definitie van Pigoski is echter wel op die eindgebruiker gericht.

De beoordeling van de onderhoudbaarheid van de code is een intern controlemiddel voor Info Support; er is geen sprake van een “oplevering bij klant”. In feite wordt er tijdens het dagelijkse buildproces telkens een nieuwe versie opgeleverd, die steeds dichter in de buurt komt van de uiteindelijke versie richting de klant.

Er kan bijvoorbeeld sprake zijn van een interne oplevering, zoals een web service, die onderdeel uitmaakt van een groter op te leveren systeem. Juist het dagelijks monitoren tijdens het ontwikkelproces, waarin een indicatie wordt gegeven over hoe moeilijk het is om aanpassingen op zo’n

afgerond component door te voeren, is interessant. Indien dit relatief laag blijft, dan wordt de kans groter dat de software op tijd wordt opgeleverd aan de klant. Ook wordt de kans groter dat na oplevering aan de klant aanpassingen sneller worden uitgevoerd.

Volgens Van Dale betekent een oplevering “overdracht van het voltooide aangenomen werk”. Omdat de doelgroep van de indicator gericht is op een technisch medewerker, betreft het ook het werk dat door hem wordt uitgevoerd. Werk kan gezien worden als een (kleine) taak, die door een ontwikkelaar wordt uitgevoerd, zoals het implementeren van een functie.

In dit onderzoek wordt voor softwareonderhoud de definitie van IEEE gehanteerd, waarbij het echter wel om de oplevering van werk van de ontwikkelaar aan de manager betreft. Dit komt enigszins ook overeen met de gedachte van Pigoski, dat onderhoud begint voor oplevering aan klant.

Onderhoudbaarheid van code

De onderzoeksvraag richt zich echter op onderhoudbaarheid en het is dus van belang om duidelijk vast te stellen wat hieronder verstaan wordt. Een manier om onderhoudbaarheid scherper te stellen is door het op te splitsen in verschillende aspecten.

Aspecten volgens Karlsson

Karlsson heeft een onderhoudsmodel specifiek voor code opgesteld (Karlsson, 1995) met daarin de volgende vijf aspecten:

Aspect	Voorbeeld
Consistentie	Zijn er eenduidige guidelines, die consequent worden toegepast?
Zelfbeschrijvendheid	Komt tijdens het lezen de intentie van de code duidelijk naar voren?
Simpelheid	Is de code makkelijk te begrijpen?
Modulariteit	In hoeverre is functionaliteit gescheiden in de code?
Testbaarheid	In hoeverre kan getest worden of de aanpassingen geen verstoringen elders in het systeem veroorzaken?

Tabel 4 Aspecten volgens Karlsson

Tussen de aspecten is enige overlap te onderkennen. Zeker het aspect simpelheid (tegenovergestelde van complexiteit) heeft invloed op andere aspecten.

Door consistent te coderen kan aangenomen worden dat er voor de lezer minder afleiding ontstaat om het daadwerkelijke probleem te begrijpen dat het codefragment oplost, want er wordt telkens dezelfde structuur gehanteerd. In die zin kan aangenomen worden dat consistentie bijdraagt aan een verlaging van de complexiteit.

Ook zelfbeschrijvendheid zorgt ervoor dat code beter te begrijpen is. Zo staan In het boek Code Complete voorbeelden genoemd hoe code geschreven kan worden waarin de intentie duidelijk is en waarbij het regels commentaar beperkt blijft (McConnell, 2004).

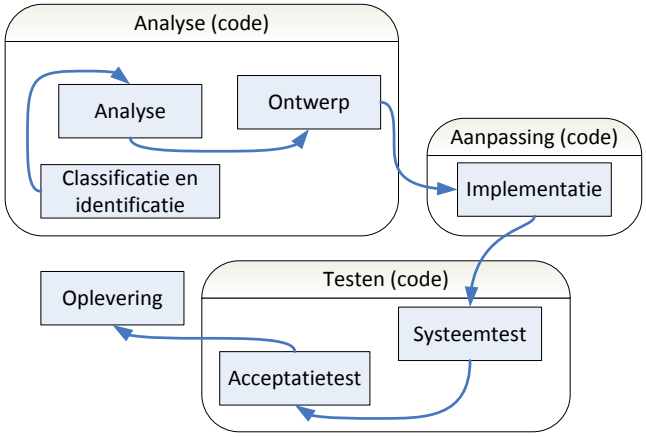
Bij modulariteit wordt dezelfde functionaliteit afgebakend in aparte modulen. Bij het lezen van de code hoeven de hersenen per probleem zo min mogelijk informatie tegelijkertijd te verwerken, waardoor de complexiteit daalt.

Voor het meten van onderhoudbaarheid is het niet wenselijk als de afzonderlijke aspecten elkaar overlappen, want dan zijn deze niet meer bij elkaar op te tellen. Dit is noodzakelijk om op basis van verschillende aspecten tot een enkele meetwaarde voor onderhoudbaarheid te komen.

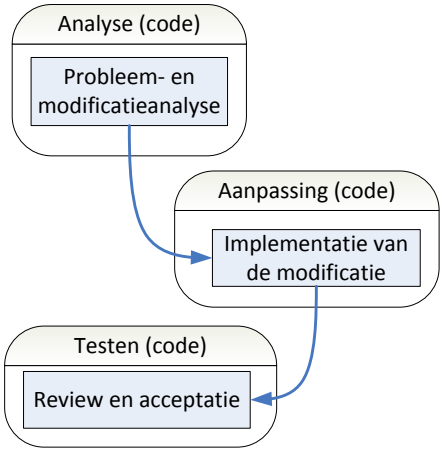
Aspecten volgens onderhoudproces

Een andere insteek is om het onderhoudsproces zelf onder de loep te nemen, waarin gekeken wordt naar de verschillende fasen.

In Figuur 5 staan de activiteiten van onderhoudsproces weergegeven volgens IEEE (IEEE, 1998) en in Figuur 6 staan deze volgens ISO (ISO, 2000). In beide figuren zijn dezelfde groepering te onderkennen: analyse, aanpassing en testen.



Figuur 5 Activiteiten uit IEEE1219-98



Figuur 6 Activiteiten uit ISO/IEC 14764-00

Op basis hiervan kan onderhoudbaarheid worden onderverdeeld in deze drie aspecten, zoals is weergegeven in Tabel 5.

Aspect	Toelichting
Analyseerbaarheid	Het gemak waarmee bepaald kan worden hoe de aanpassing precies uit te voeren. Indien er sprake is van correctief onderhoud, dan is hierbij het traceren van de fout inbegrepen.
Aanpasbaarheid	Het gemak waarmee de aanpassing doorgevoerd kan worden. Denk bijvoorbeeld aan het aantal plekken en regels code die bij de aanpassing betrokken zijn.
Testbaarheid	Het gemak waarmee gecontroleerd kan worden dat het systeem (nog steeds) volgens de specificaties werkt.

Tabel 5 Aspecten van onderhoudbaarheid op basis van het onderhoudsproces

Deze opdeling van onderhoudbaarheid is in overeenstemming met het model van Boehm, dat al was weergegeven in Figuur 3.

Verschillende soorten onderhoud

Swanson was een van de eersten die onderzocht heeft wat er precies gebeurt in evolutie en onderhoud van software, door empirische data vanuit de industrie onder de loep te nemen. Hij heeft daarbij drie verschillende categorieën van onderhoud onderkend (Swanson, 1976), welke later zijn opgenomen in IEEE- standaard (IEEE, 1993) en ISO-standaard (ISO, 2000) voor softwareonderhoud.

Soort onderhoud	Toelichting
Correctief	Aanpassen van software na oplevering om ontdekte fouten te corrigeren.
Adaptief	Aanpassen van software na oplevering om het uitvoerbare programma bruikbaar te houden in een veranderde of veranderende omgeving.
Perfectief	Aanpassen van software na oplevering om performance of andere kwaliteitsattributen te verbeteren.

Tabel 6 Drie soorten onderhoud volgens Swanson

Deze soorten onderhoud zijn ook terug te herleiden naar de definitie van softwareonderhoud volgens IEEE, die al eerder benoemd was. Naast de drie soorten onderhoud van Swanson wordt ook preventief onderhoud onderkend in beide standaarden.

Soort onderhoud	Toelichting
Preventief	Aanpassen van software na oplevering om potentiële fouten te detecteren en corrigeren, voordat het daadwerkelijke fouten worden.

Tabel 7 Preventief onderhoud als extra soort onderhoud

Bij preventief onderhoud wordt er dus pro-actief gereageerd, want er wordt ingegrepen voordat de fout zich echt voordoet. Fowler heeft voor objectgeoriënteerde software meerdere “refactorings” beschreven (Fowler, 1999). Code wordt hierbij opgeschoond zonder de functionaliteit te veranderen. Hoewel hierbij geen fouten in het programma worden voorkomen, kan er toch gesproken worden van preventief onderhoud. Het zorgt namelijk voor beter gestructureerde code, waardoor aangenomen kan worden dat toekomstige onderhoudstaken sneller uitgevoerd worden.

In de ISO-standaard worden de verschillende soorten onderhoud weer opgedeeld in categorieën (zie Tabel 8). Preventief en correctief wordt gezien als correcties van reeds aanwezige fouten; perfectief en adaptief als een verbetering van het softwareproduct beschouwd worden.

	Correctie	Verbetering
Pro-actief	Preventief	Perfectief
Re-actief	Correctief	Adaptief

Tabel 8 Indeling van soorten onderhoud in categorieën

Het punt dat perfectief altijd pro-actief is lijkt niet te kloppen. Zo kan het voorkomen dat een beveiligingsprobleem pas tijdens productie boven water komt of dat de performance van een applicatie pas na langer gebruik aan het voetlicht komt. Het is natuurlijk gewenst om deze pro-actief te onderscheppen, maar dit is natuurlijk niet altijd het geval.

Definitie van onderhoudbaarheid

Aan het begin van dit hoofdstuk is al de definitie van softwareonderhoud volgens IEEE gegeven. Hieronder staat de definitie van onderhoudbaarheid volgens IEEE, die hiermee nauw samenhangt.

Onderhoudbaarheid is het gemak van het aanpassen van een softwaresysteem of component na oplevering om:

- **fouten te corrigeren,**
- **performance (of andere attributen) te verbeteren**
- **aan te sluiten op een veranderde omgeving. (1993)**

Eerste stap richting definitie voor onderhoudbaarheid

Het woord “gemak” is echter nog een te breed begrip; zeker als deze straks gemeten moet worden. Ook is het woord gemak wellicht misleidend, omdat het suggereert dat het niet zoveel moeite kost.

Beter is om te praten over de bestede tijd, die benodigd is om een aanpassing door te voeren. Bovendien moet deze bestede tijd uitgezet worden tegen de complexiteit van de taak. Stel dat voor twee aanpassingen dezelfde tijd is besteed, maar dat de complexiteit van de taak verschillend is, dan was de onderhoudbaarheid ook verschillend.

De factoren die invloed hebben op de complexiteit van de taak zijn afhankelijk van het soort onderhoud dat gepleegd moet worden, zoals is weergegeven in Tabel 9.

Soort onderhoud	Factoren van invloed op complexiteit van de taak
Correctie (Correctief + Preventief)	• De complexiteit van het op te lossen probleem • De grootte van het te onderhouden programma • De ondersteuning van debugging tools om de fout op te sporen
Verbetering (Adaptief + Perfectief)	• De hoeveelheid nieuwe functionaliteit dat toegevoegd moet worden. • De ondersteuning vanuit de softwarearchitectuur.

Tabel 9 De complexiteit van de taak is afhankelijk van het soort onderhoud dat gepleegd moet worden

Het lijkt ook aannemelijk dat het aspect analyseerbaarheid een grotere rol speelt bij het uitvoeren van correcties dan bij verbeteringen. De bestede tijd zit meestal in het opsporen van het probleem en niet in de aanpassing zelf. Het aspect aanpasbaarheid lijkt echter weer een grotere invloed te hebben bij het verbeteren dan bij het corrigeren van de software, omdat bijvoorbeeld de architectuur hiervoor wel of geen ondersteuning biedt.

Ook de geschiktheid van de onderhoudsprogrammeur speelt een belangrijke rol. In het boek Code Complete wordt op basis van onderzoeken gesteld dat er qua oplostijd factor 20-tot-1 verschil zit tussen ervaren en onervaren programmeurs (McConnell, 2004).

Een factor die hierbij een rol speelt is de intelligentie van de persoon. Ook de reeds opgedane ervaring en kennis over de te onderhouden applicatie en gebruikte technologie spelen een rol.

Verder is aangegeven in deze scriptie dat oplevering wordt gezien als het werk van een werknemer ten behoeve van zijn manager; dus niet de oplevering van het softwarehuis aan de klant.

Ook komt in de definitie van IEEE komt de scheiding tussen functionele en niet-functionele aanpassingen niet expliciet aan bod.

Eerdergenoemde punten heeft geleid tot de onderstaande definitie voor onderhoudbaarheid.

Onderhoudbaarheid is de bestede tijd per taakcomplexiteit per geschiktheid van de (onderhouds)programmeur om een softwaresysteem of component aan te passen na oplevering om:

- **fouten (vroegtijdig) te corrigeren,**
- **nieuwe functionaliteit toe te voegen. (functioneel)**
- **kwaliteitsattributen te verbeteren. (niet-functioneel)**

Het betreft hierbij de oplevering van werk, dat uitgevoerd is door een (onderhouds)programmeur ten behoeve van zijn opdrachtgever.

Gehanteerde definitie voor onderhoudbaarheid in deze scriptie

Conclusie

Onderhoudbaarheid opgesplitst in de volgende aspecten:

- analyseerbaarheid
- aanpasbaarheid
- testbaarheid

Verder is onderhoudbaarheid scherpgesteld in een nieuwe definitie.

Vanaf nu wordt in deze scriptie met “onderhoudbaarheid” de nieuwe definitie bedoeld.

4. Metrieken met betrekking tot onderhoudbaarheid

In het vorige hoofdstuk is het begrip onderhoudbaarheid scherp gesteld, waarmee de eerste deelvraag is beantwoord. Dit hoofdstuk beantwoordt de tweede deelvraag uit de probleemstelling, welke luidt:

Welke metrieken met betrekking tot onderhoudbaarheid kunnen op basis van statische analyse van objectgeoriënteerde code verkregen worden?

Hierbij is ook de volgende subdeelvraag opgesteld:

Wat betekenen specifieke meetwaarden van deze metrieken, waar liggen de grenswaarden?

Codemetriek, meting en meetwaarde

Softwaremetrieken is volgens Fenton et al. een algemene term voor het beschrijven van een breed scala activiteiten, die benodigd zijn om metingen te verrichten in het veld van de software engineering. Deze activiteiten variëren van het produceren van getallen die eigenschappen van broncode karakteriseren tot modellen die helpen met het voorspellen van benodigde middelen en softwarekwaliteit (Fenton, et al., 2000). Het bepalen welke metingen uitgevoerd gaan worden is hiervan slechts een onderdeel.

In deze scriptie wordt zich beperkt tot code. Hieronder is de relatie tussen codemetriek, meting en meetwaarde uitgelegd, zoals er in deze scriptie tegenaan gekeken wordt.

Een *codemetriek* is een eigenschap van de code. De input van een *meting*[1] is code, dat als output een *meetwaarde* oplevert (dat de grootte aangeeft van die metriek).

Relatie tussen codemetrieken, metingen en meetwaarden

Selectie van de metrieken

Het doel is om metrieken te vinden, die een relatie hebben met onderhoudbaarheid en op basis van statische analyse uit objectgeoriënteerde code onttrokken kan worden.

In een overzichtspaper zijn alle tot dan toe bekende onderzoeken gepresenteerd, waarin de relatie tussen objectgeoriënteerde metrieken en kwaliteit is onderzocht (Briand, et al., 2002). Dit bevat dus ook alle onderzoeken met betrekking tot onderhoudbaarheid, dat een deelaspect is van kwaliteit.

De twee algemene metrieken Lines of Code (LOC) en Cyclomatic Complexity (CC), zijn niet specifiek voor objectgeoriënteerde code zijn, maar wordt wel vaak gebruikt. LOC is een algemene gebruikte maat voor de omvang van een systeem en CC zegt iets over het aantal beslispunten in een programma.

Van alle uitgevoerde onderzoeken komen metrieken uit de verzameling van Chidamber en Kemerer (Chidamber, et al., 1994) regelmatig terug. Vanwege deze populariteit is deze verzameling gekozen.

Uit een experiment bleek dat de drie metrieken Interaction Level, Operation Argument Complexity en Interface Size bruikbaar waren voor het voorspellen van de onderhoudbaarheid (Bandi, et al., 2003).

[1] Een formelere beschrijving van een meting volgt in Hoofdstuk 5, waarin de meettheorie wordt besproken.

Vanwege de beperkte tijd zijn andere metrieken verder buiten beschouwing gelaten. De geselecteerde metrieken staan weergegeven in Tabel 10.

Groep	Metriek
Algemeen	Lines of Code (LOC) Cyclomatic Complexity (CC)
Chidamber en Kemerer	Depth of Inheritance (DIT) Number of Children (NOC) Coupling Between Objects (CBO) Response For A Class (RFC) Lack of Cohesion of Methods (LCOM) Weighted Methods Per Class (WMC)
Bandi et al.	Interaction Level (IL) Interface Size (IS) Operation Argument Complexity (OAC)

Tabel 10 Gekozen metrieken voor dit onderzoek

Eigen analyse

De metrieken zijn per groep vanuit twee aandachtspunten “berekening” en “interpretatie” geanalyseerd, zoals staat toegelicht in Tabel 11. Eventueel wordt vooraf nog relevante informatie gegeven over de metriek.

Aandachtspunt	Toelichting
Berekening	Hierin wordt uitgelegd hoe de metriek berekend kan worden, zodat er vanuit een eenduidige definitie verder beredeneerd kan worden.
Interpretatie	Hierin wordt uitgelegd wat een verhoging van de meetwaarde tot gevolg heeft voor onderhoudbaarheid. Hierbij wordt aangenomen dat overige factoren (zoals bijvoorbeeld de functionaliteit) gelijk blijven. Per aspect van onderhoudbaarheid (analyseerbaarheid, aanpasbaarheid en testbaarheid) wordt uitgelegd of de verschuiving positief en/of negatieve gevolgen heeft.

Tabel 11 De aandachtspunten waarop de metrieken geanalyseerd zijn

Algemeen

Cyclomatic Complexity (CC)

Aandachtpunt	Uitwerking		
Berekening	$Cyclomatic\ Complexity = aantal\ binaire\ beslispunten + 1$ Indien het om een drievoudige beslissing gaat, dan wordt deze geteld als een twee binaire beslissingen; een case statement met n alternatieven wordt geteld als n-1 binaire beslispunten.		
Interpretatie	Analyseren	-	Indien het aantal paden toeneemt, dan moeten deze ook geanalyseerd worden (vooral bij correctief onderhoud).
	Aanpassen	-	*Geen invloed op aanpasbaarheid. Verhoging van complexiteit zorgt ervoor dat de (probleem)analyse moeilijker wordt, maar zegt niks of het gemak van het aanpassen zelf.*
		+	
	Testen	-	Indien het aantal paden toeneemt, dan moeten deze ook getest worden.

Tabel 12 De berekening en interpretatie van de metriek Cyclomatic Complexity (CC)

Lines Of Code (LOC)

Een algemeen gebruikte meetwaarde voor de grootte van de broncode is Lines of Code(LOC). Er zijn echter verschillende manier om LOC te berekenen, want dit is afhankelijk van hoe een regel geteld wordt. Denk hierbij aan hoe de volgende punten:

- Lege regels code
- Regels commentaar
- Declaraties van data
- Regels met meerdere instructies

Jones rapporteert dat sommige tellingen wel een factor vijf groter zijn dan andere, simpelweg vanwege verschillende technieken om te tellen (Jones, 1986).

Er is een soort van algemene consensus dat lege en commentaarregels niet geteld worden (Fenton, et al., 1997), omdat dit geen invloed heeft op het uiteindelijk gecompileerde programma.

Aandachtpunt	Uitwerking		
Berekening	$LOC = Totale\ aantal\ regels\ code - aantal\ lege\ regels\ code - aantal\ regels\ commentaar$		
Interpretatie	Analyseren	-	Indien de LOC toeneemt, dan moeten er meer regels gelezen worden, zodat de analyseerbaarheid afneemt.
		+	Stel dat alle functionaliteit in een enkele klasse gestopt is. Als deze vervolgens wordt gescheiden over verschillende klassen, dan kan het aantal regels toenemen, terwijl deze wel beter te begrijpen is.
	Aanpassen	-	LOC zegt niets over de structuur van de code, dus het is niet aan te geven welke impact dit heeft op de aanpasbaarheid.
		+	
	Testen	-	Indien de LOC toeneemt, dan moeten er meer regels getest worden, zodat de testbaarheid afneemt.
		+	Stel dat alle functionaliteit in een enkele klasse gestopt is. Vervolgens worden deze gescheiden over meerdere klassen en er worden interfaces gedefinieerd, waarop getest kan worden. Hierdoor zijn de testen flexibeler geworden, omdat ze afhankelijk zijn van een algemene interface in plaats van concrete implementaties.

Tabel 13 De berekening en interpretatie van de metriek Lines of Code (LOC)

Chidamber en Kemerer

Depth of Inheritance Tree (DIT)

Aandachtpunt	Uitwerking		
Berekening	Het aantal niveaus van de betreffende klasse tot aan de wortel van de overervingboom. Als de klasse van meerdere klassen tegelijk erft, dan geldt het aantal niveaus van het langste pad. In Java en C# is er echter geen ondersteuning voor “multiple inheritance”.		
Interpretatie	Analyseren	-	Meer klassen hebben potentieel impact op die bepaalde klasse. Denk aan extra overgeërfde attributen en methoden van bovenliggende klassen.
		+	Het kan extra informatie geven over hoe klassen zich tot elkaar verhouden (bijvoorbeeld: elk hond is een dier).
	Aanpassen	-	Als meer overerving gebruikt wordt, dan is de kans groter dat bij een aanpassing een nieuwe klasse geïntroduceerd moet worden om toch consistent te blijven aan het ontwerp
		+	Vanwege het mogelijke hergebruik van instantievariabelen en methoden in de bovenliggende klassen, kan codeduplicatie voorkomen worden.
	Testen	-	Bij het testen van een bepaalde klasse moeten ook rekening worden gehouden met alle overgeërfde attributen en methoden.
		+	Als instantievariabelen en methoden worden hergebruikt, dan hoeft deze maar een enkele keer getest te worden.
	Opmerking: Bij een overervingboom is de semantiek van cruciaal belang. Het is aannemelijk dat een semantisch logische overerving positief werkt op onderhoudbaarheid en dat een semantisch onlogische overerving negatief werkt op onderhoudbaarheid. Deze semantiek is echter niet terug te vinden in de metriek.		

Tabel 14 De berekening en interpretatie van de metriek Depth of Inheritance (DIT)

Number of Children (NOC)

Aandachtpunt	Uitwerking		
Berekening	Het aantal klassen die (in)direct erven van een bepaalde klasse.		
Interpretatie	Analyseren	-	Meer onderliggende klassen worden mogelijk beïnvloed door die bepaalde klasse.
		+	Zie om dezelfde reden “ DIT, analyseren, +”.
	Aanpassen	-	Zie om dezelfde reden “ DIT, aanpassen, -”.
		+	Vanwege het mogelijke hergebruik van instantievariabelen en methoden in de onderliggende klassen, kan codeduplicatie voorkomen worden.

Interpretatie	Testen	-	Zie om dezelfde reden " DIT, testen, -".
		+	Zie om dezelfde reden " DIT, testen, +".

Tabel 15 De berekening en interpretatie van de metriek Number of Children (NOC)

Coupling Between Objects (CBO)

Aandachtpunt	**Uitwerking**		
Berekening	Het aantal niet-geërfde koppelingen met andere klassen voor de betreffende klasse. Wanneer een methode gedeclareerd in een bepaalde klasse gebruik maakt van een methode of variabele gedefinieerd in een andere klasse, dan is er sprake van een koppeling.		
Interpretatie	Analyseren	-	Als er meer afhankelijkheden naar andere klassen zijn, dan wordt het analyseren doorgaans moeilijker, omdat telkens van klasse naar klasse "gesprongen" moet worden.
		+	Als de gemaakte koppeling tot gevolg heeft dat daardoor verantwoordelijkheden tussen klassen beter gescheiden wordt, dan kan het analyseren juist makkelijker worden. Voorbeeld: het opnemen van constante globale variabelen in aparte klasse zorgt voor extra koppelingen naar die klasse. Vanwege de variabelenaam (bijvoorbeeld KORTINGSFACTOR in plaats van hardgecodeerde waarde) is het duidelijker wat er bedoeld wordt en hoeft indien nodig maar op die enkele plek gewijzigd te worden.
	Aanpassen	-	Als er meer afhankelijkheden zijn, dan wordt de kans groter dat een aanpassing in een bepaalde klasse ook invloed heeft op andere klassen, zodat die wellicht ook weer aangepast moeten worden.
		+	De reden van de koppeling kan zijn dat functionaliteit beschikbaar komt in een nieuwe klasse, die vervolgens hergebruikt kan worden door andere klassen. Hierdoor neemt codeduplicatie af, zodat de aanpasbaarheid beter wordt.
	Testen	-	Als complexiteit toeneemt (zie "analyseren, -"), dan wordt het testen ook moeilijker.
		+	Als codeduplicatie afneemt (zie "aanpassen, +"), dan wordt de testbaarheid beter.

Tabel 16 De berekening en interpretatie van de metriek Coupling Between Objects (CBO)

Response For a Class (RFC)

Aandachtpunt	Uitwerking
Berekening	De verzameling methoden die potentieel uitgevoerd kunnen worden ten gevolge van een ontvangen bericht door een object van een bepaalde klasse, waarbij niet verder wordt gegaan dan één niveau van nesting (Chidamber, et al., 1994). De volgende aannames worden hierbij gemaakt: • Een bericht is een methodeaanroep. • Een klasse waaruit de methodeaanroep plaatsvindt heeft alleen toegang tot de publieke methoden (dus bijvoorbeeld het geval dat een afgeleide klasse wel toegang heeft tot een protected methode wordt buiten beschouwing gelaten) Op basis van deze aannames kan de volgende concretere definitie worden gehanteerd: Het aantal publieke methoden van een betreffende klasse met daarbij opgeteld het aantal verschillende methoden die direct vanuit die publieke methoden aangeroepen kunnen worden. Deze direct aan te roepen methoden kunnen via overerving of via een andere klasse bereikbaar zijn.

Interpretatie	Analyseren	-	Als er meer methoden aangeroepen kunnen worden, dan is het aannemelijk dat de klasse complexer wordt om te begrijpen, vanwege het “springen” tussen de methoden.
		+	Stel dat een bepaalde methode erg lang is, zodat deze moeilijk is om te begrijpen. Om die reden kan deze methode worden opgesplitst in kortere methoden. In dat geval neemt de RFC toe, terwijl de complexiteit van de klasse afneemt.
	Aanpassen	-	In feite zegt RFC iets over aantal mogelijke koppelpunten (publieke methoden) en de vanuit daar beschikbare koppelpunten. Koppelingen en aanpasbaarheid al besproken in “ CBO, aanpassen, -”
		+	Zie om dezelfde reden “ CBO, aanpassen, +”.
	Testen	-	Als de complexiteit toeneemt (zie “analyseren, -“), dan wordt het testen ook moeilijker.
		+	Als de complexiteit afneemt (zie “analyseren, +“), dan wordt het testen ook makkelijker.

Tabel 17 De berekening en interpretatie van de metriek Response For A Class (RFC)

Lack of Cohesion in Methods (LCOM)

Tijdens de literatuurstudie kwam naar voren dat er verscheidene definities van LCOM in omloop zijn, welke zijn weergegeven in Tabel 18. In deze scriptie is gekozen voor de definitie van Henderson-Sellers, omdat deze ondersteund werd door de tool NDepend (zie experiment later in de scriptie). Het valt buiten de scope van het onderzoek om alle verschillende soorten LCOM te beschrijven en te vergelijken.

Jaartal	Auteurs	Referentie
1991	Chidamber en Kemerer	(Chidamber, et al., 1991)
1993	Li en Henry	(Li, et al., 1993)
1994	Chidamber en Kemerer	(Chidamber, et al., 1994)
1996	Hitz en Montazerri	(Hitz, et al., 1996)
1996	Henderson-Sellers	(Henderson-Sellers, 1996)

Tabel 18 Verschillende definities voor LCOM

<table>
<tr><th>Aandachtpunt</th><th colspan="3">Uitwerking</th></tr>
<tr><td>Berekening</td><td colspan="3">$$LCOM = \frac{M - (som(MF)/F)}{M - 1}$$
• M is het aantal methoden in de klasse[2]. F is het aantal instantievariabelen in de klasse.
• MF is het aantal methoden van de klasse, die een bepaalde instantievariabele gebruiken.
• Som(MF) is de som van MF voor alle instantievariabelen in de klasse.

De waarde ligt tussen 0 en 1, waarbij 0 voor de beste en 1 de slechtste cohesie staat.</td></tr>
<tr><td>Interpretatie</td><td>Analyseren</td><td>-</td><td>Stel een klasse met vele “get”- en “set”-methoden, die alleen de bijbehorende instantievariabele gebruiken. Hierdoor scoort de LCOM-waarde beduidend slechter, want de methoden gebruiken geen overeenkomstige instantievariabelen. Toch hoeft dit niet zo’n grote invloed te hebben op de analyseerbaarheid, zeker als deze methoden netjes gegroepeerd zijn.</td></tr>
<tr><td></td><td></td><td>+</td><td>Als de methoden meer gebruik maken van dezelfde instantievariabelen, dan kan dit duiden op meer functionele samenhang. Een klasse is makkelijker te begrijpen als de methoden en instantievariabelen een gemeenschappelijk doel hebben.</td></tr>
<tr><td></td><td>Aanpassen</td><td>-</td><td>Stel dat functionaliteit, die bij elkaar hoort, versnippert over meerdere klassen. In dat geval betekent dat de kans groter wordt dat aanpassingen ook over meerdere klassen gaan, zodat het aanpassen moeilijker gaat.</td></tr>
</table>

[2] Straks in het experiment worden voor C# statische en instantiemethoden geteld, dit bevat ook constructors, properties getters/setters en events add/remove methoden (NDepend, 2007).

Interpretatie	Testen	-	Stel dat functionaliteit versnippert over meerdere klassen. In dat geval moeten er meer objecten geïnstantieerd worden voor de tests.

Tabel 19 De berekening en interpretatie van de metriek Lack of Cohesion Of Methods (LCOM)

Weighted Methods Per Class (WMC)

Aandachtpunt	**Uitwerking**		
Berekening	De som van de complexiteit van de methoden in een bepaalde klasse. Complexiteit van een methode is door Chidamber en Kemerer niet specifieker gedefinieerd[3].		
Interpretatie	Analyseren	-	Indien de totale complexiteit van de methoden toeneemt, dan kost het meer tijd om de klasse te analyseren.
		+	Stel dat de complexiteit van een methode wordt verminderd door deze op te splitsen in aparte methoden. WMC blijft in dit geval gelijk, maar de analyseerbaarheid wordt beter.
	Aanpassen	-	*Geen invloed op aanpasbaarheid. Verhoging van complexiteit zorgt ervoor dat de (probleem)analyse moeilijker wordt, maar zegt niets over het gemak van het aanpassen zelf.*
		+	
	Testen	-	Als de complexiteit van het programma toeneemt, dan worden de testen ook complexer.

Tabel 20 De berekening en interpretatie van de metriek Weighted Methods per Class (WMC)

Bandi et al.

Hieronder volgen de metrieken Interaction Level (IL), Operation Argument Complexity (OAC) en Interface Size (IS), die elk een goede Indicator voor onderhoudbaarheid bleek te zijn (Bandi, et al., 2003).

Interaction Level (IL)

Bandi is onduidelijk over de betekenis van een interactie, welke gebruikt wordt voor de metriek Interaction Level. Hij begint op een abstract niveau de betekenis van een interactie te beschrijven.

Er is een interactie tussen twee entiteiten A en B wanneer de waarde van entiteit A direct is berekend op basis van de waarde van entiteit B of vice versa (Bandi, et al., 2003).

Abstracte definitie van interactie volgens Bandi

Vervolgens wordt een interactie concreter gedefinieerd in de context van de metriek IL.

Er is sprake van een interactie als de waarde van een data-attribuut berekend wordt op basis van de waarde van een of meerdere parameters (of vice versa). (Bandi, et al., 2003).

Concretere definitie van interactie volgens Bandi in de context van de metriek Interaction Level (IL)

[3] Straks voor het experiment is de Cyclomatic Complexity gekozen als meetwaarde voor complexiteit.

Later geeft Bandi echter aan dat de interactie niet plaatsvindt in de implementatie van de methode. Het volgende voorbeeld wordt hierbij gegeven: de waarde van een parameter wordt in de implementatie van de methode opgeslagen in een van de instantievariabelen. Desalniettemin laat het ontwerp (lees: de signatuur) van de methode toe dat ook interacties kunnen plaatsvinden met de andere instantievariabelen. Het feit of deze interactie al dan niet plaatsvindt en hoe vaak dit is, is een implementatie-issue. Interaction Level wordt gepositioneerd als een “design metric” en het feit dat het ontwerp het toestaat is wat er toe doet.

Concreet gezien betekent dit dat de parameters van een methode per definitie een interactie hebben met alle instantievariabelen. Dit levert de volgende definitie van interactie op, zoals voor dit onderzoek geïnterpreteerd is. Dit komt ook overeen met de berekeningen die Bandi heeft uitgevoerd voor zijn experiment (Bandi, et al., 2002).

Elke combinatie van een parameter met een instantievariabele is een interactie.

Eigen geïnterpreteerde definitie van interactie in de context van de metriek Interaction Level (IL)

De wegingsfactor van de parameters of instantievariabelen zijn gebaseerd op het type, zoals weergegeven in Tabel 21. Bepaalde typen hiervan zijn achterhaald voor .NET of Java, bijvoorbeeld een rechtstreekse Pointer naar het geheugen. Ook is het object File een voorbeeld van de vele klassen uit de klassenbibliotheken, die .NET en Java kennen. Om consequent te blijven moeten eigenlijk aan al die klassen een wegingsfactor toegekend worden.

Type	Wegingsfactor
Boolean	0
Character of Integer	1
Real	2
Array	3
Pointer	5
Record, Struct of Object	6
File	10

Tabel 21 Typen met bijbehorende wegingsfactoren

De sterkte van een interactie is gebaseerd op de grootte van de wegingsfactor.

De sterkte van een interactie is het product van “grootte wegingsfactor parameter” en “grootte wegingsfactor instantievariabele”.

Definitie van de sterkte van een interacrie

Nu het duidelijk is wat verstaan wordt onder “interactie” en “sterkte van de interactie” kan de metriek Interaction Level beschreven worden.

Aandachtpunt	Uitwerking		
Berekening	**Niveau**	**Berekening**	
	Methode	K1 * (Aantal interacties) + K2* (Som van sterkte van de interacties)	
	Klasse	Som van de ILs van de methoden.	
	Applicatie	Som van de ILs van de klassen.	
	De constanten K1 en K2 hebben voor de eenvoudigheid de waarde 1. Indien er meer ervaring wordt opgedaan met deze metriek, dan moeten deze waarden worden bijgesteld. Deze aanpak is consistent met de aannames van andere onderzoekers voor het telkens herzien van constanten in metriekdefinities (Chidamber, et al., 1991)		
Interpretatie	Analyseren	-	Als de interface “groter” wordt, dan wordt de scope ook breder. Hierdoor neemt de complexiteit toe en dus de analyseerbaarheid af.
		+	Stel dat een methode met een hoge cyclomatic complexity wordt opgesplitst in verschillende methoden. Hierdoor wordt de analyseerbaarheid verbeterd, terwijl de IL dan juist kan stijgen voor die klasse.
	Aanpassen	-	Als de IL toeneemt, ontstaan er meer mogelijke koppelingen tussen de parameters en instantievariabelen. Hierdoor neemt de kans toe dat een wijziging in een instantievariabelen impact heeft op meerdere plekken in de klasse.
		+	Stel dat een instantievariabele als public wordt gedeclareerd, zodat andere klassen deze rechtstreeks kunnen wijzigen (aantasting van “information hiding”). Als er vervolgens een algemene wijziging gemaakt moet worden op die instantievariabele, dan moeten alle plekken die rechtstreeks die variabele gebruiken worden aangepast. Door het opnemen van een get-methode had dit probleem opgelost kunnen worden, maar de IL stijgt dan wel voor die klasse.
	Testen	-	Als de interface “groter” wordt, dan moeten er ook meer tests geschreven worden.
		+	Door het opnemen van meerdere methoden met parameters, stijgt de IL voor die klasse. Als hierdoor functionaliteit wordt opengesteld, die anders toch al aanwezig zou zijn in de applicatie, is de kans groter dat hier beter op getest wordt.

Tabel 22 De berekening en interpretatie van de metriek Interaction Level (IL)

Operation Argument Complexity (OAC)

<table>
<tr><th>Aandachtpunt</th><th>Uitwerking</th></tr>
<tr><td>Berekening</td><td>OAC is de som van de wegingsfactoren van de parameters van een methode. Deze metriek kan ook worden geaggregeerd op klasse of applicatieniveau.</td></tr>
<tr><td>Interpretatie</td><td>
<table>
<tr><td rowspan="2">Analyseren</td><td>-</td><td>Als het aantal parameters toeneemt met daarin de complexiteit van het type verwerkt, dan wordt het moeilijker om te analyseren wat er in de methode gebeurt (met daarbij de aanname dat de parameters daadwerkelijk gebruikt worden in de implementatie van die methode).</td></tr>
<tr><td>+</td><td>Stel dat een variabele niet als parameters, maar als instantievariabele wordt opgenomen, zodat deze wel bereikbaar is in de implementatie van de methode. Hierdoor daalt de OAC, terwijl ook de analyseerbaarheid afneemt.</td></tr>
<tr><td>Aanpassen</td><td>-</td><td>Zie om dezelfde reden “ IL, aanpassen, -”.</td></tr>
</table>
</td></tr>
<tr><td>Interpretatie</td><td>
<table>
<tr><td rowspan="2">Testen</td><td>-</td><td>Als het aantal parameters toeneemt met daarin de complexiteit van het type verwerkt, dan neemt het aantal initialisaties toe voor de test cases.</td></tr>
<tr><td>+</td><td>Indien de parameters niet gebruikt worden in de implementatie, dan kan dit voor onnodige test cases zorgen.</td></tr>
</table>
</td></tr>
</table>

Tabel 23 De berekening en interpretatie van de metriek Operation Argument Complexity (OAC)

Interface Size (IS)

<table>
<tr><th>Aandachtpunt</th><th>Uitwerking</th></tr>
<tr><td>Berekening</td><td>
<table>
<tr><th>Niveau</th><th>Berekening</th></tr>
<tr><td>Methode</td><td>K3 * (Aantal parameters) +
K4 * (Operation Argument Complexity).</td></tr>
<tr><td>Klasse</td><td>Som van IS van z’n methoden.</td></tr>
<tr><td>Applicatie</td><td>Som van IS van z’n klassen.</td></tr>
</table>
Net zoals bij IL zijn ook hier aan K3 en K4 de waarde 1 toegekend.
</td></tr>
<tr><td>Interpretatie</td><td>Zie de interpretatie de metriek OAC, omdat deze nagenoeg hetzelfde betekent.</td></tr>
</table>

Tabel 24 De berekening en interpretatie van de metriek Interface Size (IS)

Bestaand onderzoek

In de reeds uitgevoerde onderzoeken zijn er vanuit verschillende interpretaties naar het begrip onderhoudbaarheid is gekeken, zodat deze lastig te vergelijken zijn. Deze komen dus niet overeen met de gehanteerde definitie van onderhoudbaarheid in deze scriptie.

Zo wordt onderhoudbaarheid opgedeeld in begrijpbaarheid, analyseerbaarheid en aanpasbaarheid (Genero, et al., 2001). Het is hierbij onduidelijk wat nu het verschil tussen begrijpbaarheid en analyseerbaarheid is. Chea et al. hebben onderzocht of metrieken gebruikt kunnen worden om de onderhoudbaarheid van webapplicaties te voorspellen (Chae, et al., 2007). Hierbij is echter enkel de bestede tijd genomen en er wordt niet verteld dat bijvoorbeeld taakcomplexiteit, geschiktheid vanprogrammeur of andere soortgelijke factoren van belang zijn om iets over onderhoudbaarheid te kunnen zeggen. Beter lijkt om dan het begrip onderhoudbaarheid helemaal niet te noemen.

Een ander voorbeeld is het onderzoek van Li en Henry, waarbij in de titel van de paper “voorspellen van onderhoudbaarheid” wordt genoemd, terwijl de onderhoudsinspanning enkel als het aantal gewijzigde regels per klasse wordt gezien (Li, et al., 1993). Dagpinar et al. beschouwt onderhoudbaarheid weer als het aantal correctieve, perfectieve, adaptieve preventieve aanpassingen (Dagpinar, et al., 2003). Meten is weten, maar men moet wel weten wat gemeten wordt.

Uit het onderzoek van Breesam bleek dat de ervaring in de objectgeoriënteerde taal van invloed was om de taak sneller uit te voeren (Breesam, 2007). Hij onderzocht hierbij metrieken met betrekking tot overerving op een groep van studenten. Er wordt echter geen openheid gegeven hoe het experiment is aangepakt, zodat bijvoorbeeld niet achterhaald kan worden hoe de semantiek van de overervingboom is beïnvloed in de verschillende voorbeelden.

Wood et al. hebben een soortgelijk experiment uitgevoerd, waarbij de informatie over de overervingboom wel openlijk wordt vrijgegeven (Wood, et al., 1999). Er waren twee versies gemaakt, waarbij een versie wel overerving gebruikte en de andere niet. Dit was bewerkstelligd door in de versie zonder overerving attributen en methoden op te nemen, die anders via overerving werden meegenomen. Over de hierdoor ontstane codeduplicatie, dat vooral nadelig is voor aanpasbaarheid, wordt geen melding gemaakt. De opdracht voor de deelnemers was om een instantie van een nieuwe klasse te maken, deze te initialiseren met standaard waarden, vervolgens te wijzigen en tot slot te tonen op het scherm. De vraag is hoe representatief deze taak is voor onderhoud op deze applicatie. Stel dat een fout moet worden opgelost in een methode, die in de versie zonder overerving ook nog eens in meerdere klassen staan. Of het toevoegen van een attribuut dat in de bovenliggende klasse moet worden opgenomen. In de versie zonder overerving moet deze dan wel in meerdere klassen worden toegevoegd. Ook in een ander experiment uitgevoerd door Harrison et al. naar de diepte van overervingbomen wordt bij de bedreigingen voor de validiteit niet vermeld dat de taak niet representatief hoeft te zijn (Harrison, et al., 2000).

De onderzoeken beperken zich tot het wel al dan niet aantonen van een relatie tussen de metriek en een specifieke interpretatie van onderhoudbaarheid. Er wordt dus geen uitspraak gedaan over waar eventuele grenswaarden liggen.

Conclusie

Verschillende metrieken zijn geanalyseerd op drie aspecten van onderhoudbaarheid (analyseerbaarheid, aanpasbaarheid en testbaarheid) zoals gedefinieerd in Hoofdstuk 3. De analyse biedt verschillende concrete voorbeelden dat een verhoging van de meetwaarde verschillende effecten kan hebben op de onderhoudbaarheid, zowel in positieve als in negatieve zin. Indien een algemeen verband dus al niet te onderkennen is, dan zijn grenswaarden ook niet te bepalen.

Bestaande onderzoeken met betrekking tot metrieken en onderhoudbaarheid lijken soms te zijn "gebiassed". Er wordt naar een gewenst onderzoeksresultaat toegewerkt: de metriek moet een relatie hebben tot onderhoudbaarheid. Hierdoor wordt de objectiviteit verloren, want bepaalde aspecten die tegenovergestelde kunnen suggereren worden niet vermeld.

Bovendien wordt onderhoudbaarheid op verschillende manieren gemeten, waardoor geen eerlijke vergelijkingen gemaakt kunnen worden. Ook zijn er twijfels of deze metingen überhaupt wel aan onderhoudbaarheid gelijkgesteld mogen worden.

Meer objectief onderzoek is dus benodigd om een uitspraak te kunnen doen over metrieken en onderhoudbaarheid. Op basis van dit hoofdstuk kan de volgende hypothese opgesteld worden:

Als in de wetenschappelijke literatuur een metriek gepositioneerd wordt als een indicator voor onderhoudbaarheid, dan lijkt dit op basis van eigen interpretatie niet altijd te kloppen. De verwachting is dat de beweringen uit de eigen interpretatie ook empirisch aangetoond kunnen worden.

Deze hypothese wordt getoetst in het experiment, dat staat beschreven in de hoofdstukken 6 t/m 9. Eerst volgt nog een hoofdstuk over de haalbaarheid van een enkele meetwaarde voor onderhoudbaarheid op basis van codemetrieken.

5. Haalbaarheid van een enkele meetwaarde voor onderhoudbaarheid

Info Support is op zoek naar een enkele meetwaarde, zodat automatisch een beoordeling over de onderhoudbaarheid van de code gemaakt kan worden. Verschillende codemetrieken moeten hiervoor worden vertaald naar zo'n enkele meetwaarde. Dit hoofdstuk geeft antwoord op de volgende deelvraag uit de probleemstelling:

Kunnen de meetwaarden van verschillende metrieken worden geconsolideerd tot een enkele meetwaarde voor onderhoudbaarheid?

Meettheorie in vogelvlucht

Om tot een enkele meetwaarde te komen, is het belangrijk om stil te staan bij wat een meting eigenlijk betekent, welke soorten metingen bestaan en welke schaal een meetwaarde kan hebben.

Definitie meting en meetwaarde

Een meting is een afbeelding van de empirische wereld naar de formele, relationele wereld (Fenton, et al., 1997).

Definitie meting volgens Fenton & Pfleeger

Een meetwaarde is een getal of symbool dat toegekend is aan een entiteit op basis van een meting om een attribuut te karakteriseren (Fenton, et al., 1997).

Definitie meetwaarde volgens Fenton & Pfleeger

Empirisch zegt iets over hoe zaken worden ervaren. Het is bijvoorbeeld duidelijk te zien dat een olifant groter is dan een tijger. "Groter dan" is dan een binaire relatie, gedefinieerd op een verzameling van paren van dieren. "Groter dan" is in dit geval een empirische relatie voor lengte. Er kunnen ook meerdere empirische relaties op dezelfde verzameling bestaan. Zo kan in het voorbeeld ook een relatie "veel groter dan" bestaan. De meeste mensen vinden een olifant veel groter dan een mug, alhoewel hier minder overeenstemming over is als bij "Groter dan", vanwege het subjectieve karakter van "veel".

Door middel van een meting wordt de empirische relatie afgebeeld naar een getal uit een numeriek systeem, dat in dit geval dus de enkele meetwaarde moet worden. Hierbij moet blijven gelden dat het gedrag van metingen in een numeriek systeem gelijk blijft aan overeenkomstige elementen in de empirische wereld. Dus als de onderhoudbaarheid hoger of lager wordt, dan moet dit ook uit de index blijken. Dit wordt ook wel de representatieconditie genoemd (Fenton, et al., 1997).

Soorten metingen

Er zijn verschillende soorten metingen te onderkennen. De belangrijkste voor een enkele meetwaarde worden toegelicht. Allereerst komt de interne en externe meting aan bod en daarna een meting voor beoordeling en voorspelling.

Interne en externe meting

Een interne metriek is een andere soort eigenschap dan een externe metriek. Een interne metriek zegt iets over de eigenschap van het product zelf. In deze scriptie gaat het om code en de beschreven codemetrieken uit Hoofdstuk 4 behoren dus tot deze categorie.

Externe metrieken vertellen echter iets over de uitwerking van het product. Onderhoudbaarheid kan dus gezien worden als een externe metriek, die (al dan niet) beïnvloed wordt door de interne codemetrieken.

Meting voor beoordeling en voorspelling

Een meting voor beoordeling vertelt iets over wat nu bestaat of wat gebeurd is in het verleden. Echter, in veel omstandigheden is het gewenst iets te meten dat nog niet bestaat, dat het geval is bij een meting voor voorspelling.

Voorspellingssysteem

Een voorspellingssysteem heeft de volgende drie kenmerken (Littlewood, 1988):

1. Wiskundig model
2. Procedures om de parameters van het model te bepalen
3. Procedures voor het interpreteren van de resultaten

Ten eerste moet er een model zijn, zodat de voorspelde waarde berekend kan worden. Deze berekening gebeurt aan de hand van bepaalde parameters, want de voorspelling moet gebaseerd zijn op bepaalde input van gegevens. Om dit model te kunnen gebruiken moet dus vaststaan hoe deze parameters verkregen kunnen worden, dat vastgelegd is in procedures. Tot slot moet duidelijk zijn hoe de resultaten geïnterpreteerd moeten worden. Denk bijvoorbeeld aan de mogelijke mate van onnauwkeurigheid.

Meten van onderhoudbaarheid

In Hoofdstuk 3 is een nieuwe definitie van onderhoudbaarheid gegeven. Gesteld kan worden dat onderhoudbaarheid iets zegt over de “bestede tijd” per “taakcomplexiteit” per “geschiktheid van programmeur” voor het maken van een aanpassing. Het betreft dus een meting voor voorspelling, omdat het om de verwachte bestede tijd gaat.

Door te meten hoelang iemand met een aanpassing bezig is, kan de bestede tijd gemeten worden. Het is echter lastiger om “taakcomplexiteit” en “geschiktheid van persoon” van een persoon te meten.

De factoren die van invloed zijn op de complexiteit van de onderhoudstaak waren al weergegeven in Tabel 9. Ook deze geschiktheid van een persoon is ook van verschillende aspecten afhankelijk, zoals de intelligentie, inzet en ervaring en zijn daarom niet in een enkele meetwaarde te omvatten.

Vanaf nu wordt in deze scriptie de “onderhoudbaarheid” gemeten door de bestede tijd, waarbij (zoveel) gewaarborgd moet worden dat de taakcomplexiteit en geschiktheid van de persoon gelijk blijven.

Bestaande kwaliteitsindices

Tijdens de literatuurstudie kwamen twee kwaliteitsindices aan het licht: Static Confidence Factor (Jansen, et al.) en de Maintainability Index (Coleman, et al., 1994). Hoewel de SFC iets zegt over betrouwbaarheid in plaats van onderhoudbaarheid, is het toch interessant om de gebruikte benadering onder de loep te nemen.

Static Confidence Factor (SFC)

Het bedrijf TIOBE Software hanteert een enkele metriek voor softwarekwaliteit, welke is beschreven in eigen gepubliceerd paper (Jansen, et al.). Met kwaliteit wordt hier de betrouwbaarheid bedoeld. Er wordt duidelijk aangegeven dat niet alle fouten op basis van statische analyse gevonden kunnen worden, dus hierover kan de SFC geen informatie verschaffen.

De paper bestaat voornamelijk uit het wiskundig onderbouwen van de SFC. De meetwaarde van de SFC is een decimaal getal tussen de 0 en 100.

In onderstaande tabel zijn de drie aspecten van een voorspellingssysteem en de validatie beschreven voor de SFC.

Aspect	Uitwerking
Wiskundig model	De onderstaande parameters zijn van belang voor de input: Codeerstandaard De codeerstandaard is een verzameling van regels. Voorbeelden van regels zijn “Voorkom het gebruik van multiple inheritance” of “Een methode mag geen hogere Cyclomatic Complexity dan twintig hebben”. Kans op defect De kans op defect geeft de ernst van een regel uit de codeerstandaard aan met betrekking tot de betrouwbaarheid van het programma. Dit is een numerieke waarde tussen de 0 en de 1. Het geeft een indicatie dat een overtreding van de regel daadwerkelijk een fout in de software oplevert. Zo heeft bijvoorbeeld de regel “gebruik returnwaarden van functies” een hogere waarde dan “gebruik altijd haakjes voor if-then-else statements”. Overtredingen Er moet bekend zijn welke regels uit de codeerstandaard in overtreding zijn voor een bepaald programma. Deze kunnen gecontroleerd worden op basis van statische codeanalyse door middel van zogenaamde code checkers. Voor de exacte onderbouwing en de afgeleide formule wordt naar de betreffende paper doorverwezen.
Procedures om de parameters van het model te bepalen	Het bepalen van de parameter “kans op een defect” speelt een cruciale rol, want dit is de koppeling tussen het resultaat van de codeanalyse en het kwaliteitsattribuut betrouwbaarheid. In de paper wordt dit echter alleen aangedragen als interessant toekomstig onderzoek.

Procedures voor het interpreteren van de resultaten	Een verhoging van de meetwaarde betekent dat het programma betrouwbaarder is geworden (op basis van de fouten die statisch gevonden kunnen worden).
Validatie	De meetwaarde is enkel op basis van ervaringen getoetst, waarbij bleek dat “de SCF helpt managers en engineers om een globale impressie van de kwaliteit van een softwareprogramma te krijgen”.

Tabel 25 De drie kenmerken van een voorspellingssysteem en validatie voor de Static Confidence Factor

Maintainability Index

De Maintainability Index (MI) is voorgesteld om de onderhoudbaarheid te bepalen op basis van de broncode(Coleman, et al., 1994) . Dit lijkt dus precies datgene te zijn, waar Info Support op zoek naar is.

Aspect	Uitwerking
Wiskundig model	De formule luidt als volgt: $$MI = 171 - 5.2 \cdot \ln(A) - 0.23 \cdot B - 16.2 \cdot \ln(C) + 50.0 \cdot \sin\sqrt{2.46 \cdot D}$$ Hierbij geldt: • A = Halstead Volume (Halstead, 1977). Dit is een samengestelde metriek gebaseerd op het aantal operatoren en operanden in de broncode. • B = Gemiddelde Cyclomatic Complexity (McCabe, 1976) per module. • C = Gemiddeld aantal regels code per module. • D = Percentage regels commentaar per module. De formule is het resultaat van het verzamelen van data van meerdere industriële softwaresystemen en beoordelingen door experts op basis van subjectieve assessments.
Procedures om de parameters van het model te bepalen	De parameters A, B, C en D kunnen allemaal op basis van statische broncodeanalyse verkregen worden.
Procedures voor het interpreteren van de resultaten	Als de waarde van MI stijgt, dan is de code beter onderhoudbaar.
Validatie	De MI is gevalideerd op basis van subjectieve evaluaties door experts. Er is in de paper weinig informatie gegeven over de gebruikte systemen (Coleman, et al., 1994). In een voorbeeld waarin de formule wordt toegepast is een systeem gebruikt dat geschreven is in de programmeertaal C voor op het UNIX platform. De vraag is natuurlijk in hoeverre dit representatief is voor de systemen die Info Support ontwikkelt, die geschreven worden in C# en Java.

Tabel 26 De drie kenmerken van een voorspellingssysteem en validatie voor de Maintainability Index

De Software Improvement Group (SIG) heeft meerdere jaren ervaring met het toepassen van de MI op allerlei verschillende systemen. Er zijn hierbij meerdere beperkingen boven water gekomen (Heitlager, et al., 2007):

- Uit ervaring van SIG blijkt dat onderhoudsproblemen zich voornamelijk voordoen in de uitschieters met exceptioneel hoge complexiteit. Deze worden niet gevonden omdat er gebruik gemaakt wordt van een gemiddelde complexiteit.
- Er bestaat geen eenduidige definitie van wat onder een operand en operator wordt verstaan in een taal als Java of C#.
- Percentage regels commentaar blijkt geen goede relatie met onderhoudbaarheid te hebben. Zo kan code zelf uitgecommentarieerd zijn. Of het commentaar is gedateerd en behoort tot code uit voorgaande versies. Bovendien kan er ook extra commentaar toegevoegd zijn, juist omdat de code zo complex is.
- De formule is moeilijk te begrijpen en uit te leggen aan stakeholders. Waarom is de cyclomatic complexity vermenigvuldigd met 0.23? Waarom staat commentaar onder een wortel in een sinusfunctie?
- Het is moeilijk te bepalen waarom de waarde daalt en welke stappen genomen moeten worden om de waarde hoger te krijgen. Hierdoor wordt moeilijker om invloed uit te oefenen door het management, zodat waarde moeilijk geaccepteerd wordt.

Conclusie Maintainability Index en Static Confidence Factor

De besproken indices Static Confidence Factor (SCF) en Maintainability Index (MI) leveren een getal op, maar wat het nu precies meet is onduidelijk. Als bijvoorbeeld de Static Confidence Factor hoger wordt, dan wordt de software betrouwbaarder. Er is echter geen harde vertaling naar het (verwachte) aantal verwachte fouten per tijdseenheid. Zolang dat er niet is, hoe kan dan gezegd worden dat de betrouwbaarheid beter wordt? Wat wordt er nu eigenlijk gemeten?

Hetzelfde geldt voor de Maintainability Index. Een hogere waarde betekent een betere onderhoudbaarheid, waarbij in de paper gerefereerd wordt naar de definitie van IEEE. Zoals is toegelicht in Hoofdstuk 3 blijft deze definitie nog vaag over wat onderhoudbaarheid precies betekent, vandaar dat deze in de scriptie is aangescherpt. De vraag blijft wat de Maintainability Index dus precies meet, want het is niet duidelijk waarin de index uit te drukken is.

Conclusie

Het bepalen van de onderhoudbaarheid is een meting voor voorspelling. Op basis van broncodeanalyse worden automatische metingen verricht, die interne meetwaarden opleveren. Samen met eventuele wegingsfactoren moet via een berekening een enkele meetwaarde voor onderhoudbaarheid worden bepaald, die een externe metriek is.

De enkele meetwaarde moet worden uitgedrukt in een bepaalde eenheid, zodat duidelijk is waarin een voorspelling wordt gemaakt. In deze scriptie is gekozen om onderhoudbaarheid in bestede tijd uit te drukken, met daarbij de aanname dat de taakcomplexiteit en geschiktheid van

onderhoudsprogrammeur gelijk zijn. Dit zijn echter wel aspecten die niet in de code terug te vinden zijn en dus als parameter in het wiskundige model beschikbaar moeten zijn.

Om überhaupt op zo'n enkele meetwaarde op basis van codemetrieken uit te komen, moet er worden aangetoond dat er een relatie ligt tussen interne codemetrieken en de externe metriek onderhoudbaarheid. Zoals in Hoofdstuk 4 al gesteld is lijkt deze er niet te zijn en in de volgende hoofdstukken wordt het bijbehorende experiment uitgebreid besproken.

6. Experiment: opzet (algemeen)

In dit hoofdstuk staat de algemene opzet voor het experiment beschreven. Het gebruikte laboratorium, deelnemers, applicaties en uitvoering worden uitvoerig besproken.

Aanleiding

Binnen software engineering zijn de volgende drie belangrijke onderzoekstypen te onderkennen (Fenton, et al., 1997):

Type onderzoek	Toelichting
Experiment	"Onderzoek in het klein" Een rigoureuze en gecontroleerd onderzoek van een activiteit, waarbij belangrijke factoren worden geïdentificeerd en gemanipuleerd om het effect op een specifieke uitkomst te bepalen. Vanwege de hoge mate van controle zijn er een klein aantal personen of gebeurtenissen betrokken.
Case study	"Onderzoek in het specifieke" Belangrijke factoren worden geïdentificeerd die invloed hebben op het resultaat van een activiteit en vervolgens wordt de activiteit gedocumenteerd: inputs, beperkingen, benodigdheden en outputs. Doorgaans is dit niet retrospectief, dus er kan invloed uitgeoefend worden op de te onderzoeken situatie en benodigde data kan vervolgens worden vastgelegd. De focus ligt op een enkel project (en niet alle mogelijke gevallen).
Survey	"Onderzoek in het groot" Op basis van een set van data (van grote groepen of projecten) wordt bepaald hoe de populatie reageerde op een specifieke methode, tool, techniek of op trends en relaties. Er wordt teruggekeken op een situatie dat zich reeds heeft voorgedaan en er is dus geen controle geweest over de onderzochte situatie.

Tabel 27 Drie typen van onderzoek binnen Software Engineering

Vanwege de gelijke omstandigheden, die benodigd zijn om onderhoudbaarheid in tijd uit te drukken, is ervoor gekozen om een experiment te gaan uitvoeren. Op deze manier wordt zo dicht mogelijk bij de gehanteerde definitie uit hoofdstuk 3 gebleven.

Aandachtspunten vooraf

Bij het opzetten van het experiment wordt extra met onderstaande punten rekening gehouden, zodat niet dezelfde fouten gemaakt worden als tijdens eerdere soortgelijke experimenten.

Openheid

Om een experiment op waarde te kunnen beoordelen, is volledige openheid ook een belangrijke doelstelling. Openheid wordt gegeven door het beschikbaar stellen van alle gebruikte documentatie en broncode in de bijlagen. Op deze manier is bovendien replicatie van het experiment mogelijk, waarmee het onderzoeksresultaat later nog versterkt of verzwakt kan worden.

De semantiek van de code wordt belicht, zodat hierover ook inzicht wordt gegeven.

Objectiviteit

In hoofdstuk 4 is al benoemd dat bestaande onderzoeken “gebiassed” lijken te zijn. Dit experiment probeert echter zo objectief mogelijk te zijn. Denk bijvoorbeeld aan het berekenen van alle metrieken, in plaats van enkel de onderzochte metriek. Bovendien kan door de eerdergenoemde openheid objectiever het experiment beoordeeld worden.

Correctheid

Tijdens een experiment moeten de deelnemers opdrachten uitvoeren, waarvan het dus belangrijk is dat deze op correctheid gecontroleerd worden. Alleen op deze manier kunnen verschillende metingen eerlijk vergeleken worden.

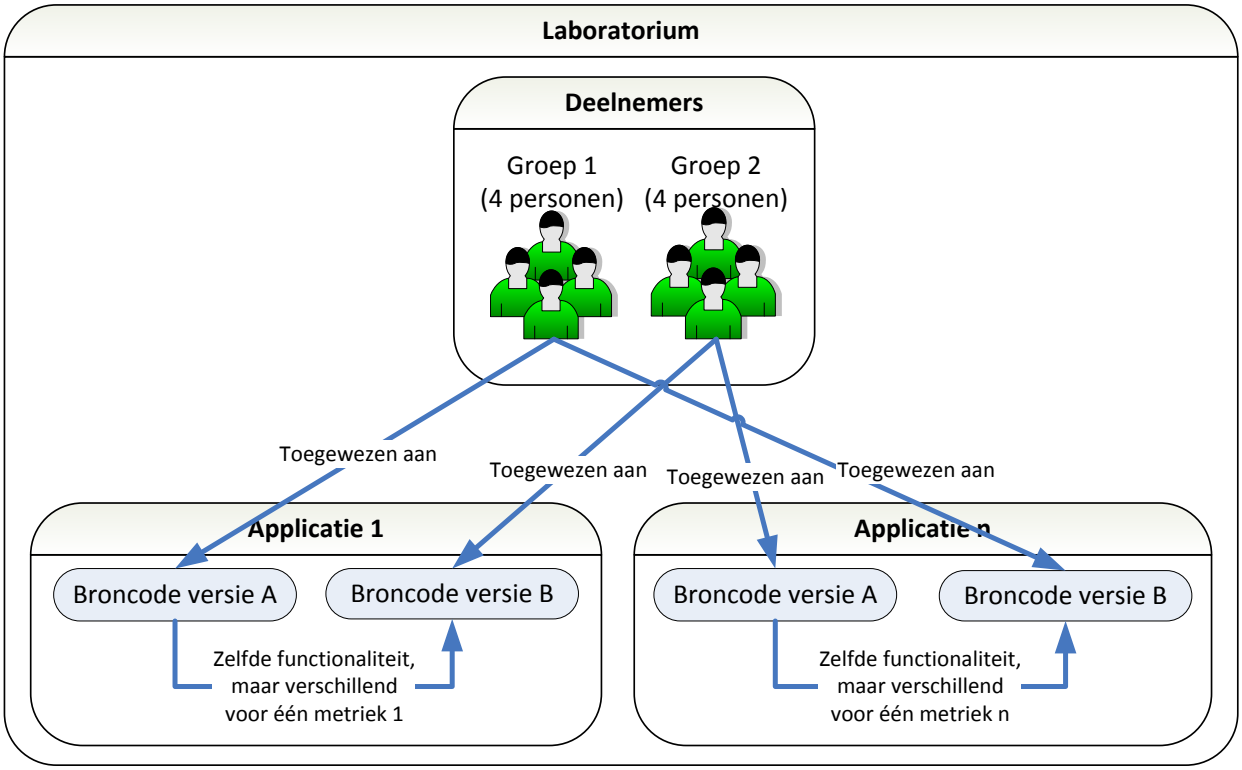
Bocco et al. beschrijft een experiment, waarin de relatie tussen interne metrieken en onderhoudbaarheid wordt onderzocht (Bocco, et al., 2005). De opdrachten zijn uitgevoerd op het niveau van een klassendiagram in UML. Het is onduidelijk hoe eenduidig op correctheid gecontroleerd is, want het betreffen hier aanpassingen op ontwerpniveau, zonder concrete implementaties op codeniveau.

Semantisch kloppend

De gebruikte structuren moeten semantisch wel kloppen. Zo gebruiken Aggarwal et al. een klasse “Result”, die erft van de klassen “InternalExam” en “ExternalExam”. De klassen “InternalExam” en “ExternalExam” erven vervolgens van klasse “Student” (Aggarwal, et al., 2006). Dit betekent dus dat een examen of een resultaat een specifiek soort student is, hetgeen onlogisch is.

Een soortgelijk voorbeeld is teruggevonden, waar een cursus erft van een student (Bocco, et al., 2005). Het lijkt dus dat dus de focus enkel op de metrieken ligt, waarbij dus voorbeeldprogramma’s worden gemaakt die semantisch onlogisch zijn. Natuurlijk worden programma’s niet altijd correct ontworpen, maar aangenomen kan worden dat dit soort voorbeelden niet representatief zijn voor objectgeoriënteerde applicaties.

Globale opzet



Figuur 7 De globale opzet van experiment met daarin de toewijzing van deelnemers aan versies

Laboratorium

Het experiment is uitgevoerd in een cursuslokaal op het kenniscentrum van Info Support, dat speciaal gereserveerd was. De ruimte bevatte acht werkstations (Intel Core 2 Duo 2.66 Ghz processor met 4 GB werkgeheugen), werkstation voor instructeur, beamer en een whiteboard.

De ontwikkelomgeving met broncode is klaargezet in een virtuele omgeving (Virtual PC), zodat de images makkelijk gedistribueerd kunnen worden over de werkstations. Bovendien kan dan gegarandeerd worden dat iedereen in exact dezelfde omgeving werkt. Er was 2 gigabyte werkgeheugen gereserveerd voor de virtuele machine.

Deelnemers

Info Support heeft een speciale toelatingsprocedure voor sollicitanten die een afstudeeropdracht willen uitvoeren. De deelnemers van het experiment behoren tot de selecte groep, die de procedure succesvol hebben doorlopen. Hieruit kan dus geconcludeerd worden dat huidige afstudeerders een bepaald kennisniveau hebben, dat een belangrijk criteriapunt is.

De deelnemers worden opgesplitst in twee groepen:

- Deelnemer 1 t/m 4 zitten in groep A
- Deelnemer 5 t/m 8 zitten in groep B

De toekenning van personen aan de deelnemernummers heeft volstrekt willekeurig plaatsgevonden.

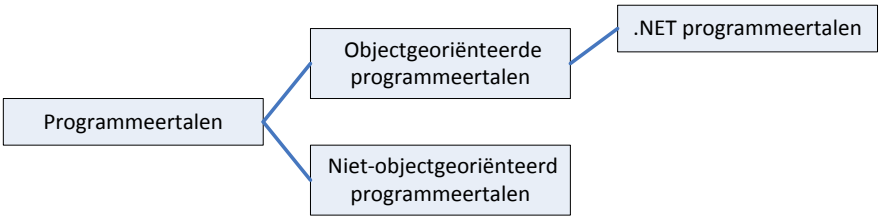
Voorafgaand aan het experiment is door de deelnemers een vragenlijst ingevuld welke als Bijlage A is opgenomen.

Het feit of iemand afstudeert voor een HBO-bachelor of een WO-master is verder buiten beschouwing gelaten in het experiment. Aangenomen kan worden dat universitaire afstudeerders gemiddeld een hogere intelligentie hebben, maar afstudeerders van een hogeschool zijn gemiddeld pragmatischer ingesteld, dat voordeel biedt tijdens uitvoeren van opdrachten.

De aanname is dat de deelnemers dezelfde capaciteiten hebben.

Na inventarisatie bleek dat de meeste deelnemers ervaring hebben met C# en ontwikkelomgeving Visual Studio. Om deze reden is C# als programmeertaal voor het experiment gekozen.

.NET is taalonafhankelijk, zodat er dus keuze is uit meerdere programmeertalen. C# is de standaard binnen Info Support. Elke .NET-taal is objectgeoriënteerd. Om een beeld te krijgen van relevante programmeerervaring is gekeken naar ervaring in .NET en programmeerervaring in het algemeen door een scheiding te maken in al dan niet objectgeoriënteerd. Deze opdeling is weergegeven in Figuur 8.



Figuur 8 Gehanteerde opdeling van programeertalen

Aan de deelnemers is gevraagd om een indicatie te geven vanaf welk jaar ze zijn begonnen en hoeveel volledige werkweken dit geweest is. Uiteraard blijft dit een ruwe schatting.

Deelnemer	Niet objectgeoriënteerd		Objectgeoriënteerd		.NET	
	Jaartal	# Weken	Jaartal	# Weken	Jaartal	# Weken
Deelnemer 1	2003	20	2001	70	2003	30
Deelnemer 2	2002	25	2001	100	2004	42
Deelnemer 3	2002	25	2001	100	2004	60
Deelnemer 4	2003	20	2003	60	2005	40
Deelnemer 5	2003	25	2004	35	2006	25
Deelnemer 6	2003	40	2004	40	2006	20
Deelnemer 7	2001	26	2001	52	2005	40

Deelnemer 8	-	0	2003	80	2005	60
Gemiddelde	2002.42	22.63	2002.25	67.13	2004.75	39.63
Mediaan	2003	25	2002	65	2005	40
Std. Dev.	0.79	11.06	1.39	25.02	1.04	14.77

Tabel 28 Ervaring programmeertalen (jaartal begonnen en hoeveelheid ervaring in werkweken)

Zoals uit Tabel 28 op te maken is liggen de jaartallen dicht bij elkaar. Er zijn wel verschillen te ontdekken in de bestede werkweken, maar gesteld kan worden dat de populatie voor het experiment bestaat uit onervaren programmeurs. Het gemiddelde aantal werkweken voor een gemiddelde softwareontwikkelaar ligt hoger ligt.

De aanname is dat de deelnemers gelijkwaardige programeerervaring hebben opgedaan.

Applicaties

De algemene opzet van een applicatie is weergegeven in Figuur 10. Een applicatie bestaat uit een opdrachtomschrijving en broncode (inclusief tests).

Opdrachtomschrijving

Beide groepen krijgen dezelfde opdrachtomschrijving voor elke applicatie, zodat de taakcomplexiteit gelijk wordt gehouden. Pfleeger en Sommerville geven aan dat enkel twintig procent van alle onderhoudsinspanningen correctief is (Pfleeger, 2001) (Sommerville, 1995). Dit betekent dus dat het merendeel van de aanpassingen in de categorie verbeteringen vallen.

Aangezien de uit te voeren aanpassing representatief moet zijn is ervoor gekozen om per applicatie enkel een verbetering uit te voeren. Aanvullende opdrachten uit de categorie correctief zijn niet opgenomen, omdat dit de opdracht per applicatie te omvangrijk zou maken. Hierdoor werd het risico te groot dat het gehele experiment ging uitlopen, want er was slechts een halve werkdag beschikbaar. De prioriteit ligt op het aantal te onderzoeken metrieken, dat meerdere geïsoleerde applicaties tot gevolg heeft.

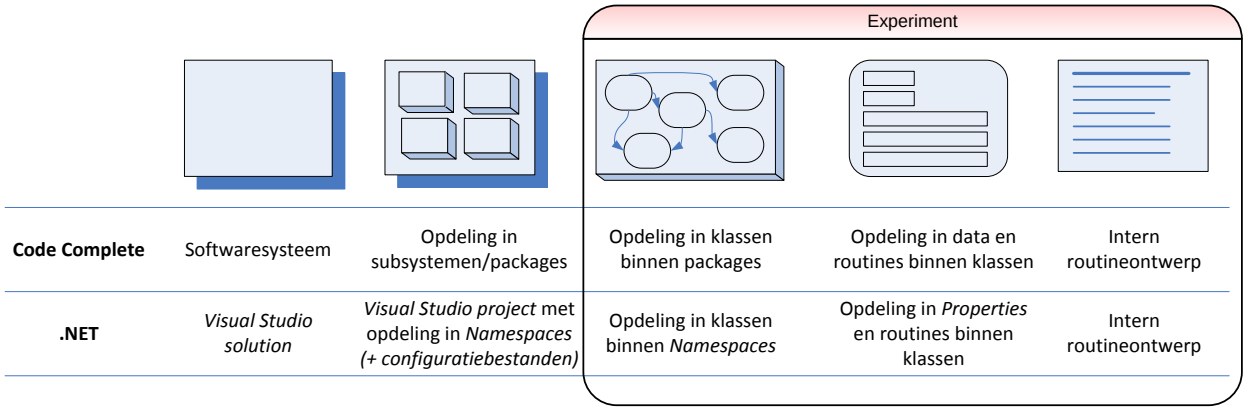
De categorie verbeteringen bestaat uit perfectief en adaptief onderhoud, zoals eerder was weergegeven in Tabel 8. Perfectief onderhoud komt niet naar voren in de aanpassingen van het experiment, omdat deze te specifieke kennis vereisen. Voor bijvoorbeeld een veiligheidsprobleem in .NET moet er kennis van specifieke klassenbibliotheken bekend zijn bij de deelnemer. Een performanceprobleem kan veroorzaakt worden door een inefficiënt algoritme of het verkeerd gebruiken van de klassenbibliotheek.

Alle opdrachten zijn dus van het soort adaptief onderhoud, zodat er dus een verandering optreedt in een bestaande situatie, waarop de software aangepast moet worden.

De opdrachten bestaan enkel uit adaptief onderhoud, zodat andere soorten onderhoud zoals correctief en perfectief niet in het experiment zijn meegenomen.

Er zijn verschillende (ontwerp)niveaus te onderkennen binnen een softwaresysteem. In het boek Code Complete is zo'n opdeling gemaakt voor objectgeoriënteerde softwaresystemen (McConnell, 2004). Deze is weergegeven in Figuur 9 tezamen met de equivalente benaming in C#/.NET, waarop dit experiment betrekking heeft.

Vanwege de beperkte tijd moeten aanpassingen niet op een te hoog niveau plaatsvinden. De grens is aangegeven in Figuur 9 , waarin te zien is dat er in het experiment niet verder wordt gegaan dan het klassenniveau.

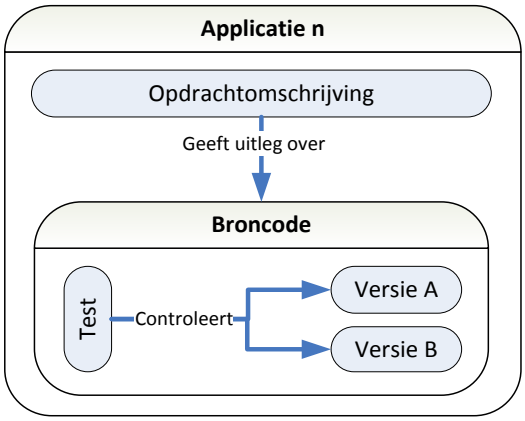


Figuur 9 Verschillende niveaus van ontwerpen volgens Code Complete en bijbehorend equivalent van .NET

Automatische tests

Dezelfde automatische tests worden klaargezet voor beide versies. Hierdoor kan automatisch en eenduidig gecontroleerd worden of de nieuwe functionaliteit correct is doorgevoerd. Door het aanleveren van een automatische test wordt werk uit handen genomen van de ontwikkelaar, want er hoeft geen test te worden opgesteld. De deelnemer is minder lang bezig, dat ook gewenst is vanwege de beperkte beschikbare tijd.

Door het aanleveren van de automatische tests wordt het aspect testbaarheid van onderhoudbaarheid verwaarloosd tijdens dit experiment.



Figuur 10 De algemene opzet van een applicatie

Twee versies

Per codemetriek wordt een aparte applicatie ontwikkeld, zodat deze volledig geïsoleerd is. Voor elke applicatie worden versies A en B gemaakt, waarbij het enige verschil moet zitten in de meetwaarde van de desbetreffende codemetriek (later blijkt dat dit moeilijk te realiseren). De applicaties hebben zoveel mogelijk dezelfde structuur en functionaliteit.

De functionaliteit van beide versies zijn hetzelfde , zodat ook de tests identiek zijn. Een uitgebreide toelichting van de structuur van de broncode volgt nog in Hoofdstuk 7, waarin te zien valt dat er getest wordt tegen dezelfde interface.

Elke deelnemer heeft een halve werkdag ter beschikking voor het experiment. Het is daarom van belang dat de broncode zo snel mogelijk te begrijpen is en niet te groot qua omvang is.

Gekozen metrieken

Om het experiment zo objectief mogelijk te houden, is gekozen voor een voorbeeld, tegenvoorbeeld en een replicatie van een bestaand experiment. In het volgende hoofdstuk wordt uitgebreid ingezoomd op de gebruikte applicaties.

Lack of Cohesion of Methods (voorbeeld)

Het voorbeeld probeert aan te tonen dat een volgens de wetenschap negatieve verandering van de metriek ook daadwerkelijk een negatief effect op de onderhoudbaarheid tot gevolg heeft.

Weighted Methods Per Class (tegenvoorbeeld)

Het tegenvoorbeeld doet een poging om juist het omgekeerde aan te tonen, waarbij een volgens de wetenschap negatieve verandering van de metriek juist een positief effect heeft op de onderhoudbaarheid.

Interface Size, Interface Level en Operation Argument Complexity (ijking)

Door het uitvoeren van een replicatie kan de gebruikte experimentopstelling voor dit onderzoek geijkt worden aan die van een bestaand experiment. Op die manier kan met meer zekerheid gezegd worden dat de gebruikte experimentopstelling correct is.

Gekozen is om het experiment van Bandi et al. te repliceren. De conclusie van dit was dat de metrieken Interaction Level, Interface Size en Operation Argument Complexity een relatie hebben met onderhoudbaarheid (Bandi, et al., 2003). Zoals straks in Hoofdstuk 7 beschreven staat, zijn de metrieken niet goed geïsoleerd. Ook heeft de “slechte” versie een betere cohesie; de “goede” versie heeft de slechtere cohesie.

Er zijn echter wel verschillen tussen de applicaties van het oorspronkelijke experiment en de replicatie:

- De broncode van het oorspronkelijk experiment was geschreven in C++. Voor de replicatie is deze herschreven naar C#, omdat hierop de focus van het onderzoek ligt. Qua semantiek van de code is niets veranderd en ook de meetwaarden van de metrieken zijn gelijk gebleven.
- Voor de replicatie is de tekst vertaald naar het Nederlands en ook qua tekststructuur zo opgezet dat deze overeenkomt met de andere twee applicaties. Bepaalde zinnen zijn scherper en eenduidiger geformuleerd. Bovendien kent het oorspronkelijke experiment twee in beperkte mate verschillende versies voor de opdrachtomschrijving, terwijl bij de replicatie voor beide versies een identieke opdrachtomschrijving wordt gebruikt. Dit om de omstandigheden voor beide versies zo constant mogelijk te houden.

Variabelen van experiment

De onafhankelijke variabele gaat uit van een oorzaak, die mogelijk invloed heeft op de afhankelijke variabele en gemanipuleerd kan worden.

De codemetrieken zijn de onafhankelijke variabelen.

De afhankelijke variabele is datgene dat gemeten wordt en waarvan verwacht wordt dat deze beïnvloed wordt door de onafhankelijke variabele.

De onderhoudbaarheid is de afhankelijke variabele, die uitgedrukt wordt in bestede tijd.

Vanwege de gelijke omstandigheden kan een verschil in de bestede tijd terug herleid worden naar het verschil in de codemetriek.

Uitvoering

De volgende expliciete keuzes zijn gemaakt voor tijdens de uitvoering van het experiment:

- Deelnemers mogen gerust vragen stellen. Indien er antwoord wordt gegeven aan de deelnemer, dan wordt deze ook medegedeeld aan de overige deelnemers.
- Deelnemers hebben enkel de opdracht om de test te laten slagen. Dit kan namelijk tijdens het experiment spoedig gecontroleerd worden. Of de aanpassing in lijn met de applicatie is en of er netjes commentaar wordt toegevoegd wordt verder niet meegenomen in dit experiment, omdat dit niet gemakkelijk te controleren is tijdens het experiment.
- Een nieuwe opdrachtomschrijving wordt uitgereikt bij aanvang of bij afronden van een applicatie. Op deze manier kan gecontroleerd worden dat de tijd correct is geregistreerd.
- Het gebruik van internet is toegestaan tijdens het experiment, omdat in een normale werksituatie dit ook beschikbaar is.

Bovendien is er voor uitvoering van het experiment een pilot uitgevoerd om te controleren of er zich geen ernstige fouten bevinden in de opdrachtomschrijvingen of broncode, die het experiment kunnen verstoren.

Conclusie

In dit hoofdstuk is de algemene opzet van het experiment besproken. Het laboratorium, deelnemers, applicaties en uitvoering met daarbij expliciet genomen keuzes zijn toegelicht. In het volgende hoofdstuk wordt ingezoomd op de broncode van de applicaties, waarbij ook de specifieke hypotheses worden opgesteld.

7. Experiment: opzet (broncode en hypotheses)

In dit hoofdstuk wordt ingezoomd op de broncode die gebruikt is tijdens het experiment, zodat hierover openheid wordt gegeven. Voor elke applicatie wordt de algehele structuur toegelicht, waarop vervolgens wordt aangewezen waar het verschil tussen de twee versies precies zit. Van de klassen, die verschillend zijn per versie, worden de meetwaarden uit hoofdstuk 4 berekend. Hieruit blijkt in hoeverre het gelukt is om de te onderzoeken metriek daadwerkelijk te isoleren van de andere metrieken. Op basis van de semantiek en de metrieken wordt een hypothese opgesteld over de onderhoudbaarheid van de broncode.

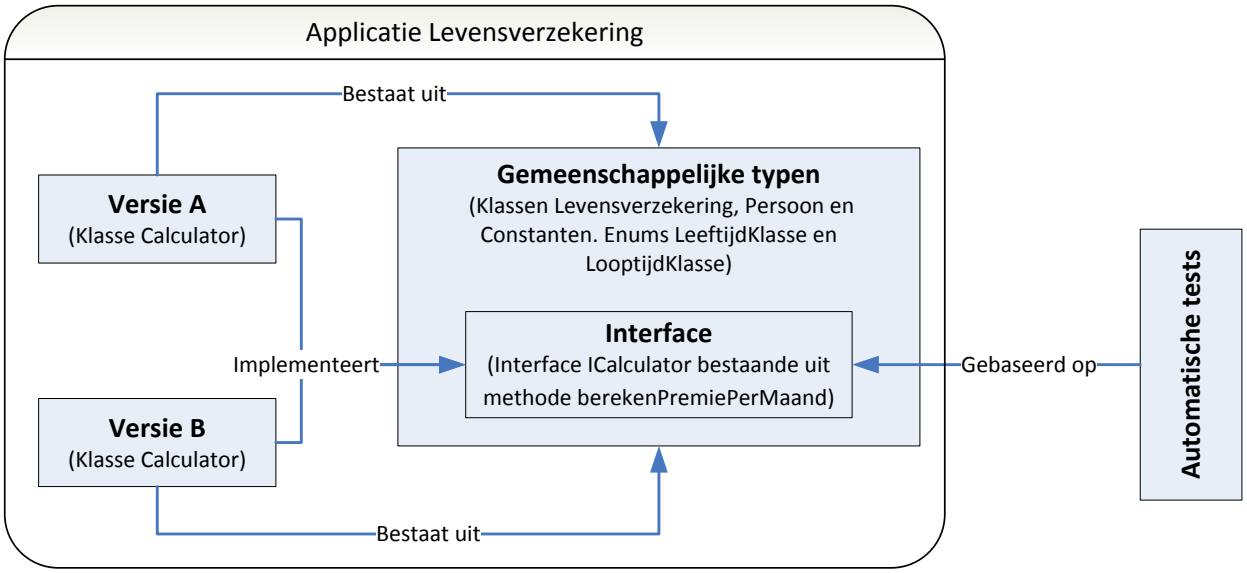
Tegenvoorbeeld: Levensverzekering (WMC)

De applicatie Levensverzekering berekent de premie per maand op basis van eigenschappen van de verzekerde persoon en de gewenste verzekering.

Doel

Het doel van deze applicatie is om aan te tonen dat een verlaging van de metriek WMC zelfs kan leiden tot een verbetering van de onderhoudbaarheid (tegenvoorbeeld).

Applicatiestructuur



Figuur 11 Structuur van de applicatie Levensverzekering

De applicatie bestaat uit algemene typen die voor zowel versie A als B hetzelfde zijn. De klassen Levensverzekering en Persoon bevatten specifieke eigenschappen, waarop de premie per maand berekend wordt. De statische klasse Constanten bevat constante waarden, die vervolgens overal in applicatie gebruikt kunnen worden. Dit biedt twee voordelen. Enerzijds kan er gerefereerd worden naar een variabelenaam in plaats van een waarde (dus ROKERSFACTOR in plaats van de waarde 1.1), zodat de betekenis duidelijker wordt. Anderzijds hoeft er bij een wijziging van de waarde van de rokersfactor maar op een enkele plek een aanpassing worden doorgevoerd, namelijk in die klasse Constanten.

Het verschil tussen beide versies zit in de klasse Calculator. Er wordt namelijk in de code op een verschillende manier de premie per maand berekend. De berekening gebeurt aan de hand van specifieke eigenschappen van de persoon en de gewenste levensverzekering, waarbij op een verschillende wijze beslissingen worden genomen worden in de code.

Beide versies van de klasse Calculator implementeren dezelfde interface, welke bestaat uit de methode berekenPremiePerMaand. De automatische tests zijn tegen deze interface geprogrammeerd, zodat identieke testcode gebruikt kan worden voor de versies A en B.

De broncode van deze applicatie en de tests kunnen teruggevonden worden in Bijlage G.

Meetwaarden

Het verschil tussen de versies zit in de klasse Calculator. Hiervan zijn de meetwaarden gepresenteerd (zie Tabel 29 en Tabel 30).

Voor alle meetwaarden in deze scriptie geldt dat LOC is berekend met de tool LocMetrics *(LocMetrics, 2007)****. DIT, NOC, LCOM en WMC zijn berekend met de tool NDepend****(NDepend, 2007)****. De metrieken RFC, CBO, IL, OAC en IS zijn handmatig berekend en terug te vinden in de Bijlagen J t/m L, omdat hiervoor geen tool gevonden kon worden.***

Klasse	LOC	DIT	NOC	RFC	CBO	LCOM	WMC	IL	OAC	IS
Calculator	73	1	0	6	22	0.5	9	63	19	25

Tabel 29 De meetwaarden voor versie A van de applicatie Levensverzekering

Klasse	LOC	DIT	NOC	RFC	CBO	LCOM	WMC	IL	OAC	IS
Calculator	103	1	0	5	48	0	24	0	12	18

Tabel 30 De meetwaarden voor versie B van de applicatie Levensverzekering

Weighted Methods Per Class (WMC)

In versie A wordt gebruik gemaakt van een multidimensionale array om op basis van huidige leeftijd en looptijd de sterftekans te bepalen, terwijl in versie B hiervoor afzonderlijke if-statements gebruikt worden. Hierdoor wordt de waarde van de Cyclomatic Complexity aanzienlijker hoger.

Overige beïnvloede metrieken

Lines of Code (LOC)

Deze metriek is gestegen, omdat er in versie B vele if-statements waren opgenomen. Dit is een voorbeeld waaruit blijkt dat metrieken niet altijd te isoleren zijn.

Coupling Between Objects (CBO)

Deze metriek is gestegen, omdat in de extra if statements van versie B telkens de leeftijd en looptijd via de get-accessor van de betreffende property wordt opgevraagd. Dit had achteraf gemakkelijk voorkomen kunnen worden door deze waarde op te slaan in een lokale variabele van die methode, maar dit is helaas pas na experiment boven water gekomen.

Lack of Cohesion of Methods (LCOM)

De LCOM waarde ligt tussen de 0 en 1, zodat een verschil van 0.5 behoorlijk groot is. Dit komt omdat de klasse Calculator uit versie A een instantievariabele heeft om de sterftekanstabel op te slaan in een tweedimensionale array, terwijl de klasse Calculator van versie B helemaal geen instantievariabelen heeft (en daardoor waarde 0 oplevert en dus zeer goede cohesie betekent). Als echter naar de code gekeken wordt, dan is er helemaal geen sprake van verschil in (functionele) cohesie: ze hebben beide exact hetzelfde doel: het berekenen van de premie per maand. De verschuiving van deze metriek wordt in dit geval dan ook genegeerd.

Interaction Level (IL)

Deze metriek is ook afhankelijk van de instantievariabelen. Aangezien versie B geen instantievariabelen heeft, blijft de waarde 0, terwijl versie A een waarde van 63 heeft.

Operation Argument Complexity (OAC) en Interface Size (IS)

Deze metrieken verschillen omdat er extra methoden zijn opgenomen in versie A, waardoor de meetwaarde stijgt.

Opdracht

Op de klasse Calculator moeten twee opdrachten worden uitgevoerd vanwege een verandering in het domein. De sterftekans is aangepast op het feit of iemand rookt. Daar bovenop wordt nog eens een rokerstoeslag berekend om het roken extra te ontmoedigen.

Deze opdracht staat uitvoerig beschreven in Bijlage D.

Hypothese

Door het gebruiken van een multidimensionale array wordt de complexiteit van de if-statements hier naartoe verschoven, terwijl de meetwaarde flink daalt, vanwege het verwijderen van die if-statements.

Voor het aanpassen van de sterftekans op versie A liggen twee alternatieven voor de hand:

- Het toevoegen van een nieuwe tweedimensionale array
- Het ombouwen van de tweedimensionale array in een driedimensionale array.

Voor het aanpassen van de sterftekans op versie B ligt het voor de hand om nieuwe if statements toe te voegen.

De verwachting is dat de probleemanalyse bij de multidimensionale arrays meer tijd kost dan bij de vele if statements, omdat die frequenter voorkomen tijdens het programmeren. Ook de aanpasbaarheid lijkt door het gebruik van multidimensionale array af te nemen, omdat er op verschillende plekken aanpassingen gemaakt moeten worden.

In tegenstelling tot wat de meetwaarden van de metriek WMC suggereren, lijkt het uitvoeren van de opdracht op versie A toch minder tijd te kosten dan op versie B.

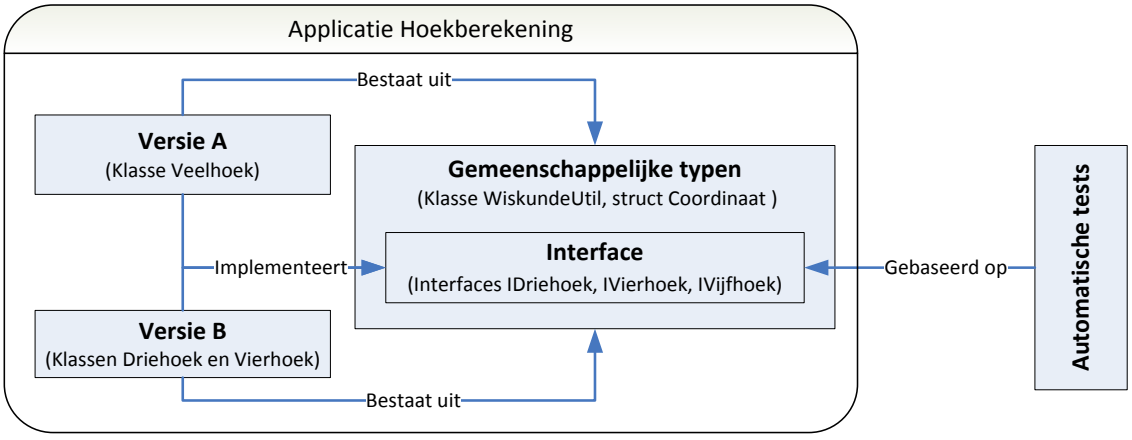
Voorbeeld: Hoekberekening (LCOM)

De applicatie Hoekberekening bepaalt op basis van de coördinaten van een veelhoek de bijbehorende hoeken.

Doel

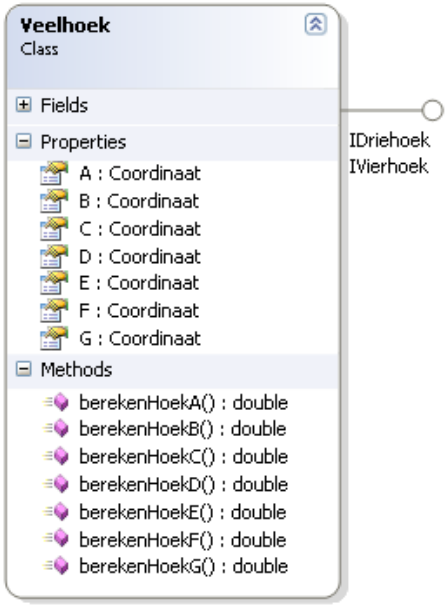
Het doel van deze applicatie is om aan te tonen dat een verlaging van de metriek LCOM daadwerkelijk kan leiden tot een verbetering van de onderhoudbaarheid.

Applicatiestructuur

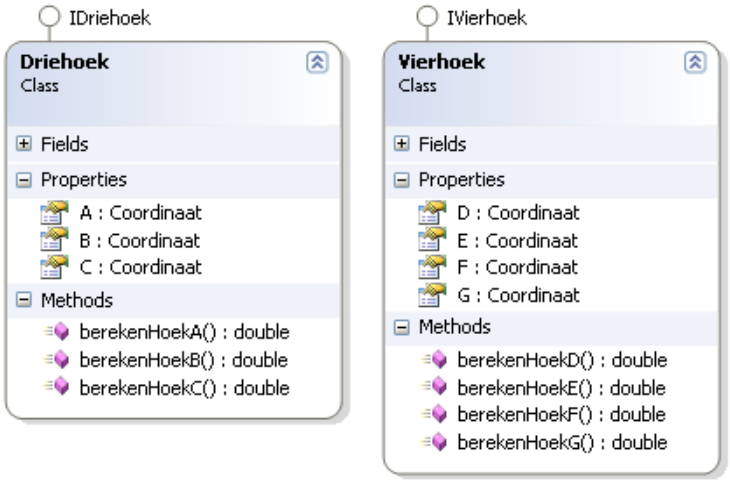


Figuur 12 Structuur van de applicatie Hoekberekening

Ook deze applicatie bestaat uit typen die voor zowel versie A als B gemeenschappelijk zijn. De klasse WiskundeUtil is statisch en hoeft dus niet geïnstantieerd te worden. WiskundeUtil biedt functionaliteit voor wiskundige berekeningen op basis van coördinaten en zijden van een driehoek.

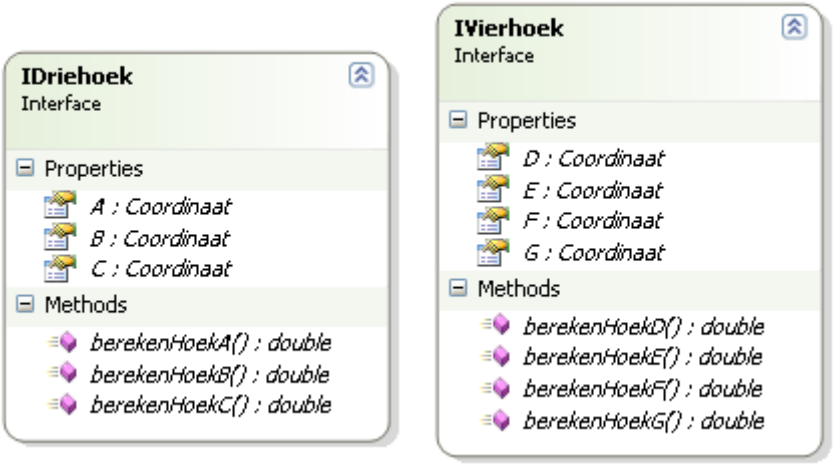


Figuur 13 Klasse veelhoek uit versie A



Figuur 14 Klassen Driehoek en Vierhoek uit versie B

Het verschil tussen versie A en B zit in enerzijds de klasse Veelhoek en anderzijds in de klassen Driehoek en Vierhoek . In Figuur 13 is te zien dat versie A een enkele klasse Veelhoek bevat, waarvan de (functionele) cohesie niet goed is, omdat deze de verantwoordelijkheden van zowel een driehoek als een vierhoek bevat. In versie B is dit verbetert door per type veelhoek een aparte klasse gebruiken, zoals weergegeven in Figuur 14.



Figuur 15 Interfaces IDriehoek en IVierhoek

De applicatie kent de interfaces IDriehoek en IVierhoek (zie Figuur 15). De klasse Veelhoek implementeert beide interfaces in versie A, terwijl in versie B de klasse Driehoek enkel de interface IDriehoek implementeert en de klasse Vierhoek enkel de interface IVierhoek.

Ook in deze applicatie zijn de automatische tests tegen de interfaces opgesteld. Door het aanbrengen van deze abstractie kan (nagenoeg) dezelfde testcode gebruikt worden voor versie A en B en daarmee ook dezelfde functionaliteit worden gegarandeerd.

De broncode van deze applicatie en de tests kunnen teruggevonden worden in Bijlage H.

Meetwaarden

Klasse	LOC	DIT	NOC	RFC	CBO	LCOM	WMC	IL	OAC	IS
Veelhoek	137	1	0	18	34	0.77	14	2450	56	70
Totaal	**137**	**-**	**-**	**18**	**34**	**-**	**14**	**2450**	**56**	**70**

Tabel 31 De meetwaarden voor versie A van de applicatie Hoekberekening

Klasse	LOC	DIT	NOC	RFC	CBO	LCOM	WMC	IL	OAC	IS
Driehoek	57	1	0	10	12	0.50	6	450	24	30
Vierhoek	89	1	0	12	22	0.56	8	800	32	40
Totaal	**146**	**-**	**-**	**22**	**34**	**-**	**14**	**1250**	**56**	**70**

Tabel 32 De meetwaarden voor versie B van de applicatie Hoekberekening

Lack of Cohesion of Methods (LCOM)

De klasse Veelhoek uit versie A heeft een hogere waarde voor de LCOM dan overeenkomstige klassen van versie B, dat een op een slechtere cohesie van de klasse duidt. Dit klopt ook, omdat er meerdere verantwoordelijkheden, namelijk de hoekberekening van een driehoek en een vierhoek, in een enkele klasse gedefinieerd zijn.

De klassen Driehoek en Vierhoek uit versie B hebben een lagere waarde dan de klasse Veelhoek uit versie A, dat op een betere cohesie van de klasse duidt. Dit klopt ook, want de verantwoordelijkheden van de Driehoek en Vierhoek zijn nu netjes gescheiden.

Overige beïnvloede metrieken

RFC (Response For A Class)

De metriek is licht gestegen. Dit komt omdat deze metriek voor een klasse het aantal unieke methoden teruggeeft. Aangezien de klassen Driehoek en Vierhoek vier dezelfde methoden gebruiken, worden deze dubbel geteld. Eigenlijk moet er een vereniging van beide verzameling methoden worden genomen en hiervan de grootte bepalen. In dat geval zijn de metrieken dan ook exact gelijk voor beide versies.

IL (Interaction Level)

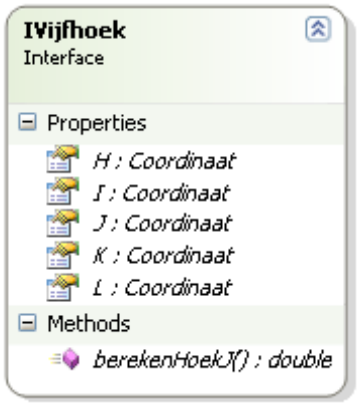
De metriek is in versie A bijna twee keer zo groot dan in versie B. Het aantal methoden blijft voor beide versies gelijk, maar in versie A heeft elke methode telkens toegang tot alle zeven coördinaten. In versie B heeft elke methode in de klasse Driehoek toegang tot de drie coördinaten en bij de klasse Vierhoek zijn er vier coördinaten beschikbaar per methode.

Bovendien valt de hoogte van beide metrieken op, dat bij zo'n relatief kleine klasse al ruim boven de duizend ligt.

Opdracht

De opdracht is om een bepaalde hoek te berekenen in een vijfhoek. De interfacedefinitie van de vijfhoek wordt hierbij al prijsgegeven aan de deelnemer (zie Figuur 16). De reden hiervoor is dat de automatische test nu tegen deze interface geprogrammeerd kan worden. Anders is het niet mogelijk om een automatische test beschikbaar te stellen, terwijl dit wel de bedoeling is, zoals in Hoofdstuk 6 beschreven is.

De gehele opdrachtomschrijving is terug te vinden in Bijlage E.



Figuur 16 De interface IVijfhoek

Hypothese

Vanwege de betere (functionele) cohesie van versie B ten opzichte van versie A, valt te verwachten dat het makkelijker is om hier nieuwe functionaliteit toe te voegen.

In overeenstemming tot wat de meetwaarden van de metriek LCOM suggereren, lijkt het uitvoeren van de opdracht op versie A meer tijd te kosten dan op versie B.

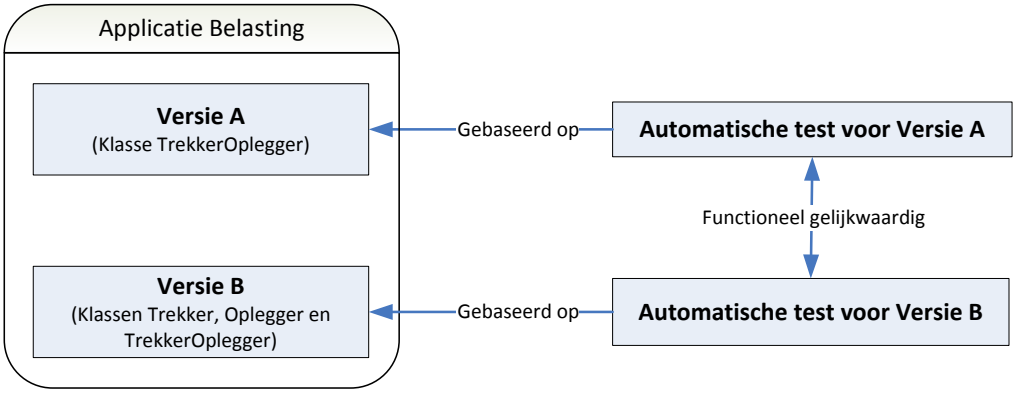
IJking: Belasting (Bandi)

De applicatie berekent de belasting op basis van eigenschappen van een oplegger en een trekker.

Doel

Het doel van deze applicatie is om de gebruikte experimentopstelling te ijken aan een eerder uitgevoerd experiment (Bandi, et al., 2003). Bovendien kan de replicatie van dit experiment het eerdere onderzoeksresultaat, dat IL, OAC en IS zijn goede metrieken voor onderhoudbaarheid, versterken of verzwakken.

Applicatiestructuur



Figuur 17 Structuur van de applicatie Hoekberekening

Versie A en B hebben geen gemeenschappelijke klassen. Versie A bestaat slechts uit een enkele klasse waarin de berekeningen van de belasting voor zowel de trekker, oplegger en de combinatie is opgenomen.

Versie B bestaat uit meerdere klassen, waarin de berekening voor de belasting van de trekker en de oplegger in gelijknamige klassen plaatsvindt. De belasting van de combinatie trekker en oplegger wordt berekend door een optelling (en maakt gebruik van andere twee klassen).

Er zijn twee verschillende automatische tests opgesteld. Dit is logisch, want er zijn ook geen gemeenschappelijke klassen of interfaces opgenomen. De tests zijn functioneel gelijk aan elkaar.

De broncode van deze applicatie en de tests zijn een rechtstreekse vertaling van de oorspronkelijke broncode, die geschreven was in C++. Aangezien C# en C++ beide objectgeoriënteerde talen zijn, kon deze vertaalslag 1-op-1 gemaakt worden.

De gebruikte broncode kan teruggevonden worden in Bijlage O.

Meetwaarden

Klasse	LOC	DIT	NOC	RFC	CBO	LCOM	WMC	IL	OAC	IS
TrekkerOplegger	112	1	0	11	0	0.74	14	236	19	33
Totaal	**112**	**-**	**-**	**11**	**0**	**-**	**14**	**236**	**19**	**33**

Tabel 33 De meetwaarden voor versie A van de applicatie Belasting

Klasse	LOC	DIT	NOC	RFC	CBO	LCOM	WMC	IL	OAC	IS
Trekker	69	1	0	8	0	0.67	9	68	11	19
Oplegger	69	1	0	8	0	0.67	9	68	11	19
TrekkerOplegger	27	1	0	7	4	0	3	188	15	19
Totaal	**165**	-	-	**23**	**4**	-	**21**	**324**	**37**	**57**

Tabel 34 De meetwaarden voor versie B van de applicatie Belasting

Interaction Level (IL)

De Interaction Level voor een methode wordt berekend op basis van zijn parameters en de instantievariabelen in die klasse. De bijbehorende wegingsfactoren hiervan, die afhankelijk zijn van het type, spelen hierbij een belangrijke rol (zie exacte berekening in Hoofdstuk 4) .

Versie B heeft een hogere totale meetwaarde voor IL dan versie A, voornamelijk omdat in de constructor van de klasse OpleggerTrekker parameters zijn meegenomen van het type Oplegger en Trekker. Ook de bijbehorende instantievariabelen zijn uiteraard van hetzelfde type Oplegger en Trekker.

Het type Object heeft een wegingsfactor van 6, terwijl een integer en double respectievelijk de weging 1 en 2 hebben (zie Tabel 21). En omdat er een vermenigvuldiging van de wegingsfactor van de parameter en instantievariabele plaatsvindt, telt een hogere wegingsfactor extra zwaar. Dit is te zien in Tabel 34, waarbij de klasse TrekkerOplegger bijna drie keer zo hoog ligt dan de klasse Trekker of Oplegger.

Het valt echter te betwijfelen of de methoden in versie B qua methodesignatuur en instantievariabele inderdaad zoveel meer complexer zijn dan in versie A. De constructor zorgt voor een uitschieter in de meetwaarde, terwijl deze niet complex lijkt. Het is enkel het opslaan van de meegegeven parameters in een instantievariabele (zie Codevoorbeeld 1) .

```
private Trekker trekker;
private Oplegger oplegger;

public TrekkerOplegger(Trekker t, Oplegger o)
{
      trekker = t;
      oplegger = o;
}
```

Codevoorbeeld 1 De Interface Level wordt relatief hoog in Versie B, omdat in de constructor de parameters en instantievariabelen van het type Object (=hoge wegingsfactor) zijn.

Hoewel de meetwaarde voor IL van versie B hoger ligt dan versie A, lijkt de semantiek van de code niet complexer. Dit in tegenstelling tot de presentatie van de gegevens in (Bandi, et al., 2003), waarbij versie B als de complexe code wordt beschouwd. Dit is op basis van de meetwaarden van de metrieken, zonder een blik op de semantiek van de code te werpen.

Operation Argument Complexity (OAC) en Interface Size (IS)

De OAC van een methode is simpelweg de som van de wegingsfactoren per parameter; de IS is de som van de OAC en aantal parameters. Net zoals bij IL geldt ook hier dat de parameters van de constructor van de klasse TrekkerOplegger uit versie B deze meetwaarden extra omhoog drukken. Ook is de methode berekenBelasting nu zowel in de klasse Trekker als in Oplegger opgenomen, dus worden de parameters dubbel geteld ten opzichte van versie A.

Overige beïnvloede metrieken

Behalve DIT en NOC zijn alle gemeten metrieken beïnvloed. Gesteld kan worden dat de Bandi-metrieken niet goed geïsoleerd zijn.

LOC (Lines of Code)

Deze metriek is gestegen, omdat de functionaliteit van een klasse verspreid is over meerdere klassen.

RFC (Response For A Class)

Versie A verschilt van Versie B, omdat functionaliteit van de oplegger en trekker in drie klassen zijn opgenomen. Algemene methode-aanroepen zoals berekenBelasting, toonGegevens en Console.Out.Write komen nu in elk van de klassen voor, zodat het totaal van versie B boven versie A ligt. Zoals al uitgelegd voor de applicatie Levensverzekering zou eigenlijk een vereniging van de verzameling methoden gepast zijn, zodat het verschil tussen beide versies aanzienlijk kleiner wordt, omdat dan alle unieke methode-aanroepen van een versie worden geteld.

CBO (Coupling Between Objects)

Versie A bestaat uit een enkele klasse en heeft dus geen koppelingen naar andere klassen. In versie B heeft alleen de klasse TrekkerOplegger koppelingen naar de klassen Trekker en Oplegger. Door het opsplitsen in aparte klassen ontstaan extra koppelingen en dit lijkt een typisch voorbeeld waaruit blijkt dat metrieken niet altijd te isoleren zijn.

LCOM (Lack Cohesion of Methods)

De meetwaarde voor LCOM geeft aan dat de (functionele) cohesie van de klassen in versie B iets beter is dan in versie A. Zeker als ook naar de semantiek van de code wordt gekeken, dan kan vastgesteld worden dat verantwoordelijkheden beter gescheiden zijn in versie B door het gebruiken van afzonderlijke klassen. Dit kan een ongewenst effect hebben op het experiment, want de versie met de slechtere Bandi-meetwaarden heeft een betere cohesie; de versie met de betere Bandi-meetwaarden heeft een slechtere cohesie.

WMC (Weighted Methods Per Class)

Elke method heeft minimaal een Cyclomatic Complexity van 1. Aangezien Versie B meer methoden bevat dan versie A, ligt de meetwaarde voor WMC ook hoger.

Opdracht

De opdracht is om twee aanpassingen uit te voeren, omdat de berekening van de belasting veranderd is. Deze gehele opdrachtomschrijving is te vinden in Bijlage F.

Hypothese

Aangezien het om een replicatie gaat van een bestaand experiment, wordt eenzelfde resultaat verwacht. De volgende hypothese kan worden opgesteld:

Net zoals de uitkomst van het oorspronkelijke experiment, kost het meer tijd om de opdracht uit te voeren op versie B dan op versie A.

8. Experiment: uitvoering en resultaten

Pilot

Indien er zich fouten in de opdrachtomschrijving of de broncode bevinden, dan kan dit het experiment verstoren. De deelnemers zijn slechts een halve werkdag beschikbaar en (onnodig) uitloop kan ontbrekende resultaten tot gevolg hebben. Om dit risico te beperken is een pilot uitgevoerd, waarin twee personen het experiment gingen “proefdraaien”. Deze personen waren ook afstudeerders bij Info Support en behoorden uiteraard niet tot de deelnemers van het echte experiment.

Onderschepte problemen

Tijdens het proefdraaien hebben zich een aantal problemen voorgedaan. De onderstaande aanpassing is de belangrijkste geweest.

Probleem	Ondernomen actie
``Assert.Fail("Verwijder deze regel, uncomment deze methode en vul bovenaan jouw klassenaam in die interface IVijfhoek implementeert"); //IVijfhoek vijfhoek; // = new JOUW_KLASSENAAM()```	```//TODO: Uncomment onderstaande code en // instantieer object van type IVijfhoek //IVijfhoek vijfhoek = new ...```
Codevoorbeeld 2 Fragment uit de testcode, zoals aanwezig tijdens het proefdraaien.	Codevoorbeeld 3 Fragment uit de testcode (aangepast), zoals aanwezig tijdens echte experiment
De testcode duwde de proefdeelnemer teveel in een bepaalde oplossingsrichting, want JOUW_KLASSENAAM suggereert het maken van een nieuwe klasse, terwijl dit helemaal niet de bedoeling was.	Dit is opgelost door JOUW_KLASSENAAM niet te benoemen in de testcode. Er is voor gekozen om de code uit Codevoorbeeld 2 te vervangen voor de code zoals weergegeven in Codevoorbeeld 3.

Het belangrijkste onderschepte probleem tijdens pilot met bijbehorende actie

De overige onderschepte problemen en genomen acties zijn opgenomen in Bijlage B.

Resultaten van pilot

De bestede tijd in minuten is per deelnemer en applicatie weergegeven in Tabel 35 en Tabel 36.

	Proefdeelnemer 1
Levensverzekering (A)	37
Hoekberekening (B)	33
Belasting (A)	23

Tabel 35 Bestede tijd van proefdeelnemer 1

	Proefdeelnemer 2
Levensverzekering (B)	36
Hoekberekening (A)	30
Belasting (B)	21

Tabel 36 Bestede tijd van proefdeelnemer 2

Het aantal bestede minuten liggen zeer dicht bij elkaar. Als dit tijdens het echte experiment zich ook voor doet, dan lijken de gestelde hypotheses niet aangenomen te kunnen worden.

Geconcludeerd kan wel worden is dat de opdrachten zelfstandig doorgewerkt konden worden. Bovendien lukte dit in de beschikbare tijd van een halve werkdag, dus dit bood vertrouwen dat het experiment niet ging uitlopen.

Voorbereiding

Foto 1 Het laboratorium is klaar voor gebruik

Een dag voor het experiment stond het cursuslokaal in het Kenniscentrum van Info Support al ter beschikking. De image is voor de tijd naar een externe harde schijf gekopieerd, zodat deze getransporteerd kon worden naar het Kenniscentrum. Via een automatisch batchproces (waarvan Info Support elke dag gebruik maakt bij een cursus) kon de image naar alle werkstations gedistribueerd worden. Vervolgens moesten vanuit afzonderlijke images de juiste bronbestanden worden klaargezet, want er zijn van elke applicatie dus twee versies beschikbaar. Er deden zich nog enkele problemen voor (zoals internettoegang vanuit de image), zodat uiteindelijk toch bijna de gehele werkdag nodig is geweest om het laboratorium klaar voor gebruik te maken (zie Foto 1).

Uitvoering

Tijdens de uitvoering van het experiment deden zich een paar speciale situaties voor die extra aandacht vereisen:

- De tests van voorbeeldapplicatie Hoekberekening slaagden initieel allemaal. Dit was in tegenstelling met de opdrachtformulering, waarin uitdrukkelijk vermeld stond dat de test eerst faalt. Dit werd opgemerkt door een deelnemer en vervolgens is dit kenbaar gemaakt aan alle deelnemers van het experiment. De reden van deze fout was de aanpassing uit Codevoorbeeld 3, waarna de Asset.Fail niet meegenomen was.

Foto 2 Opmerking bij applicatie hoekberekening

- Er was ’s ochtends een afmelding van een deelnemer in verband met ziekte. Dit is opgelost door op een andere dag het experiment voor hem te herhalen. Hij (=deelnemer 8) heeft hierbij dezelfde instructies gekregen als de deelnemers in het laboratorium.
- Alle opdrachten waren door iedere deelnemer binnen de gestelde tijd opgelost. Sommige opdrachten leken individueel uit te lopen. De verklaringen zijn opgenomen in de persoonlijke vragenlijsten, waarop later in de analyse verder wordt ingegaan.
- Bij aanvang van de gezamenlijke pauze was niemand bezig met een opdracht. De deelnemers waren klaar of ze moesten beginnen aan een volgende opdracht.

Resultaten

Bestede tijd

De resultaten van de bestede tijd worden ten eerste per groep gepresenteerd. Op deze manier kan straks geanalyseerd worden hoe de deelnemers in elke groep presteerden, zodat bijvoorbeeld verschillen qua snelheid aan het licht komen.

Ten tweede wordt de bestede tijd per applicatie in kaart gebracht, waarbij de verschillen tussen de versies bestudeerd kunnen worden.

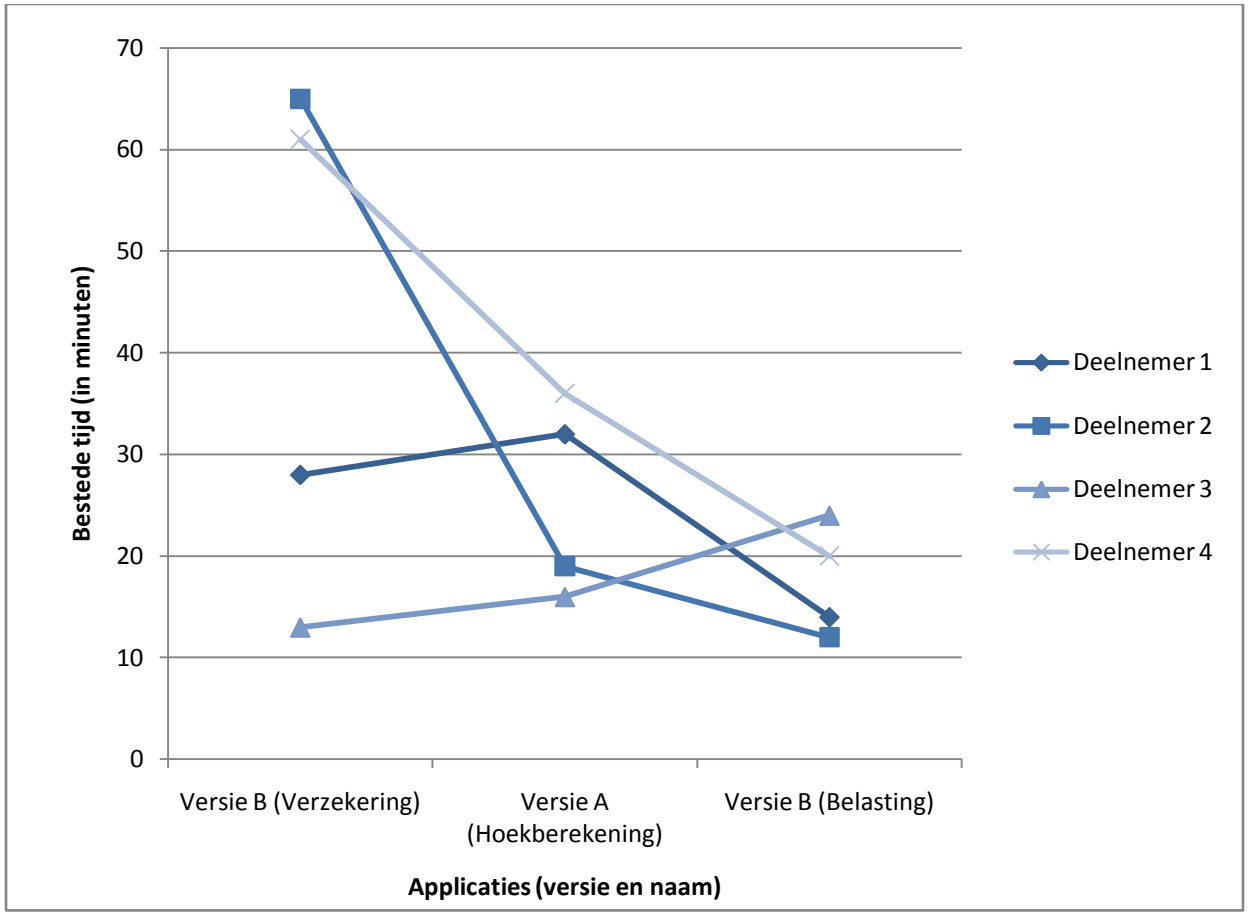
Bestede tijd per groep

Voor de prestaties per groep worden de bestede minuten per applicatie per persoon getoond.

Groep A

	Deelnemer 1	Deelnemer 2	Deelnemer 3	Deelnemer 4
Levensverzekering (Versie B)	28	65	13	61
Hoekberekening (Versie A)	32	19	16	36
Belasting (Versie B)	14	12	24	20

Tabel 37 De bestede tijd in minuten per deelnemer uit Groep A over de verschillende applicaties

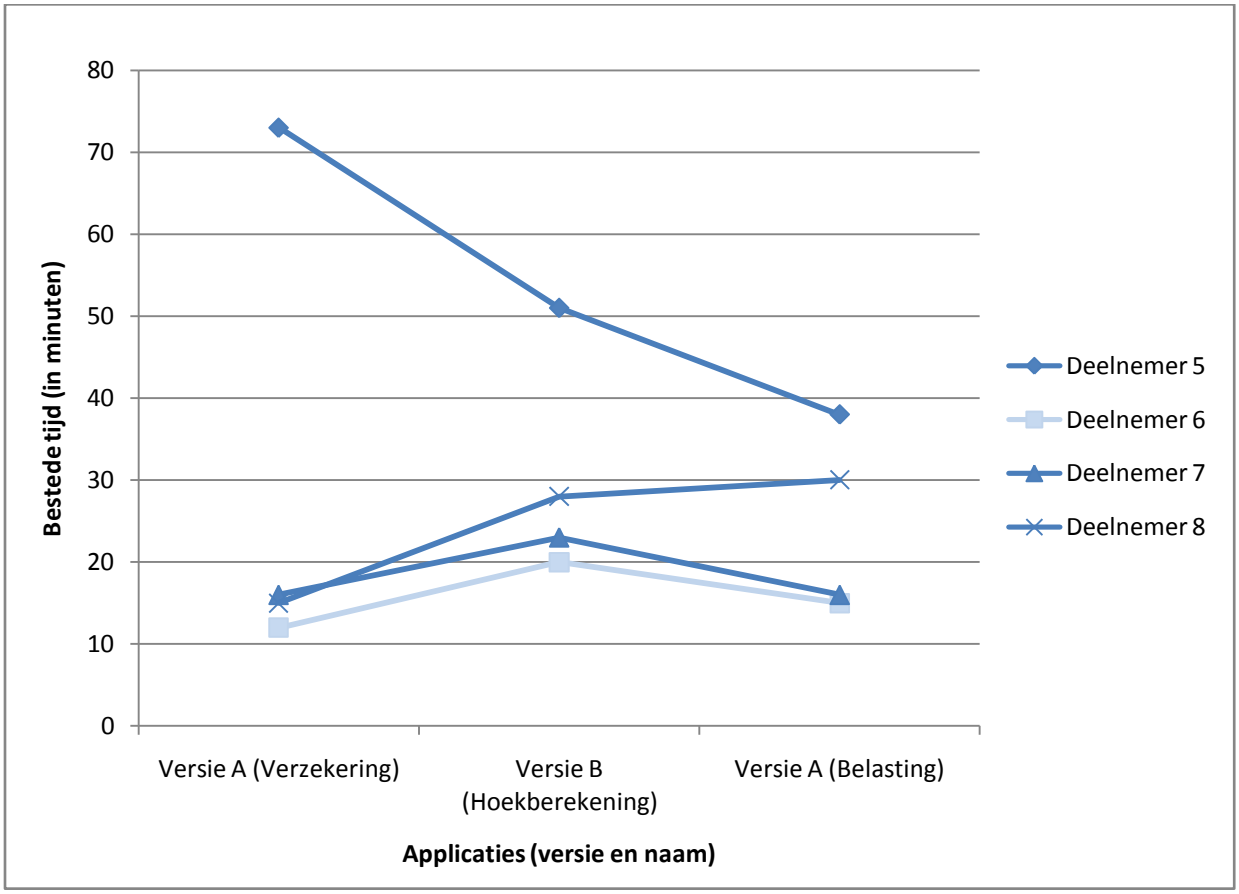


Figuur 18 Grafische weergave van de bestede tijd in minuten per deelnemer uit groep A over de verschillende applicaties

Groep B

	Deelnemer 5	Deelnemer 6	Deelnemer 7	Deelnemer 8
Levensverzekering (Versie A)	73	12	16	15
Hoekberekening (Versie B)	51	20	23	28
Belasting (Versie A)	38	15	16	30

Tabel 38 De bestede tijd in minuten per deelnemer uit Groep B over de verschillende applicaties



Figuur 19 Grafische weergave van de bestede tijd in minuten per deelnemer uit groep B over de verschillende applicaties

Bestede tijd per applicatie

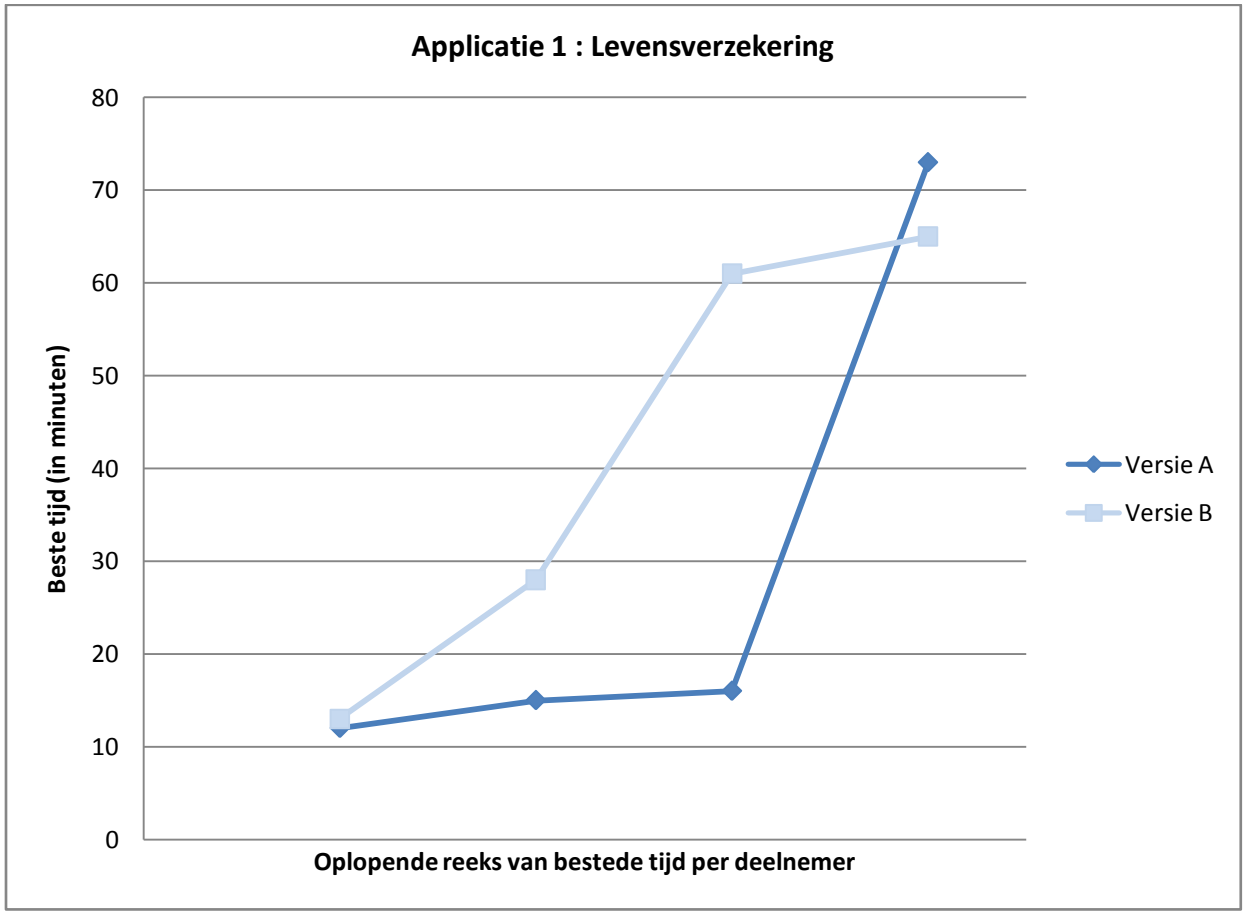
De bestede tijd per applicatie wordt gepresenteerd als een oplopende reeks. Er wordt dus begonnen met de laagste tijdmeting en daarop volgend komt telkens een hogere tijdsmeting. Op deze manier kunnen beide applicaties vergeleken worden, omdat ze op een gelijke manier gerangschikt zijn.

Ook wordt per applicatie het gemiddelde (Gem), mediaan (Med) en standaarddeviatie (Std Dev) gegeven, inclusief het verschil hiertussen (Δ).

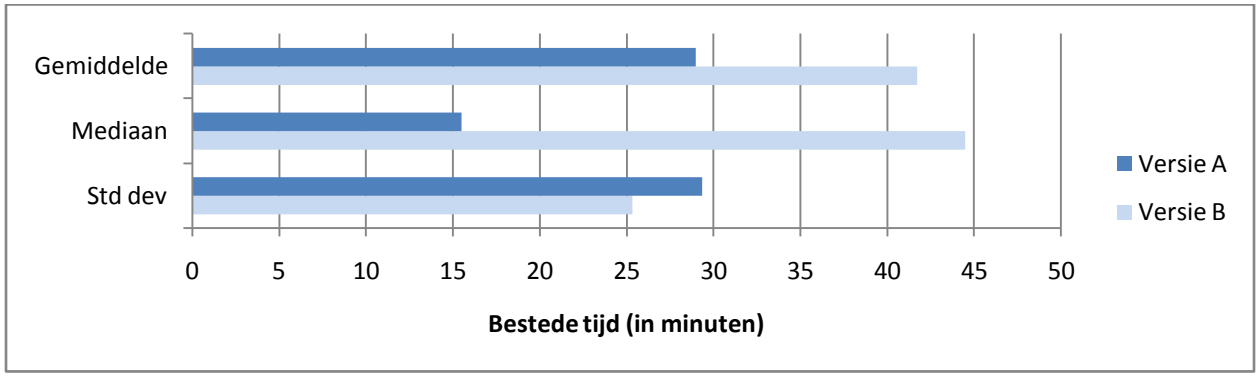
Tegenvoorbeeld: Levensverzekering (Weighted Methods Per Class)

	Oplopende reeks				Gem	Med	Std dev	Δ Gem	Δ Med	Δ Std dev
Levensverzekering (Versie A)	12	15	16	73	29	16	29	-13	-29	+4
Levensverzekering (Versie B)	13	28	61	65	42	45	25			

Tabel 39 De bestede tijd in minuten voor de applicatie Levensverzekering



Figuur 20 Oplopende bestede tijd per versie voor beide versie van Levensverzekering

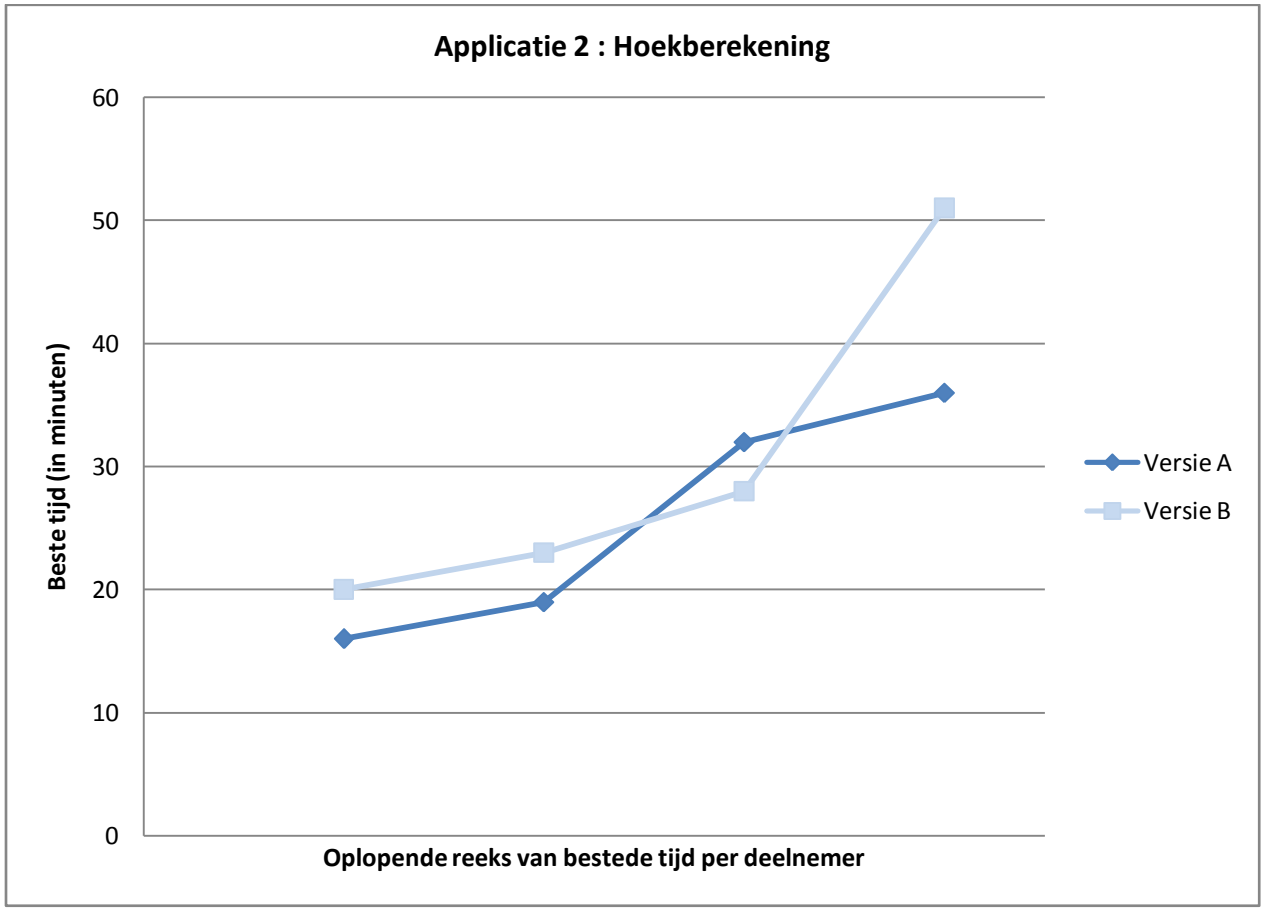


Figuur 21 Het gemiddelde, mediaan en standaarddeviatie voor beide versies van Levensverzekering

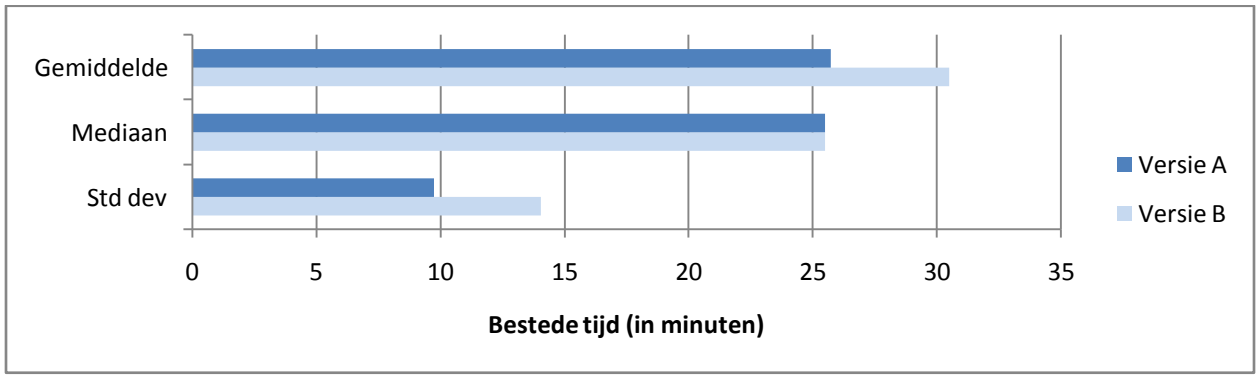
Voorbeeld: Hoekberekening (Lack Cohesion Of Methods)

	Oplopende reeks				Gem	Med	Std dev	Δ Gem	Δ Med	Δ Std dev
Hoekberekening (Versie A)	16	19	32	36	26	26	10	+5	0	-4
Hoekberekening (Versie B)	20	23	28	51	31	26	14			

Tabel 40 De bestede tijd in minuten voor de applicatie Hoekberekening



Figuur 22 Oplopende bestede tijd per versie voor beide versie van Hoekberekening

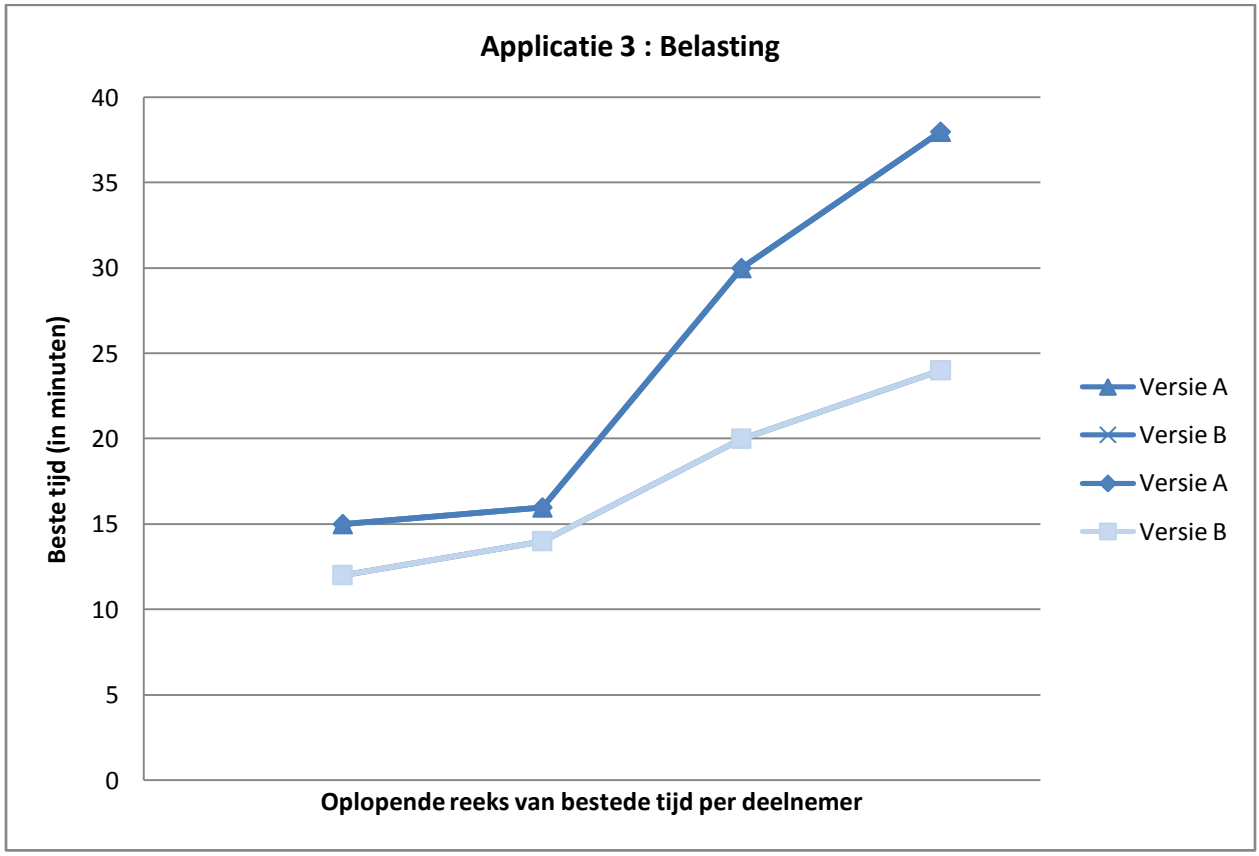


Figuur 23 Het gemiddelde, mediaan en standaarddeviatie voor beide versies van Hoekberekening

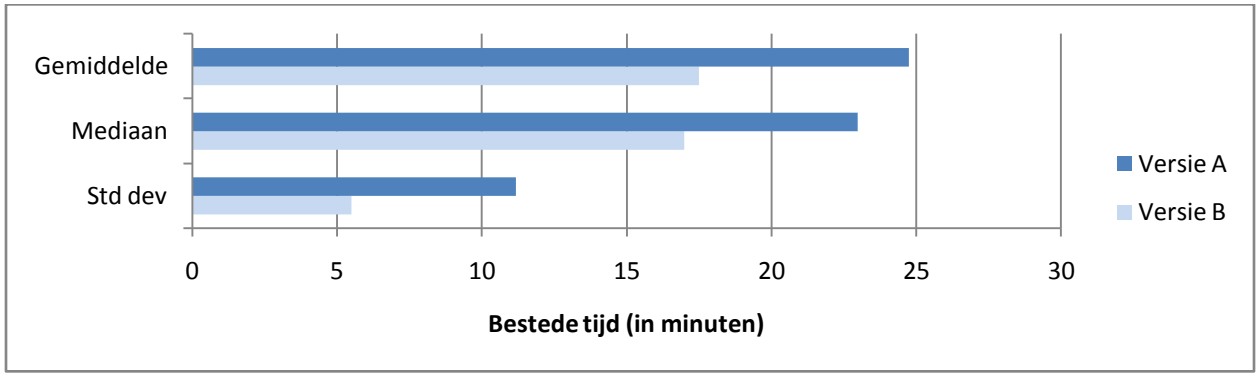
IJking: Belasting (Bandi)

	Oplopende reeks				Gem	Med	Std dev	Δ Gem	Δ Med	Δ Std dev
Belasting (Versie A)	15	16	30	38	25	23	11	-7	-6	-5
Belasting (Versie B)	12	14	20	24	18	17	6			

Tabel 41 De bestede tijd in minuten voor de applicatie Belasting



Figuur 24 Oplopende bestede tijd per versie voor beide versie van Belasting



Figuur 25 Het gemiddelde, mediaan en standaarddeviatie voor beide versies van Belasting

Ingevulde vragenlijsten

Na het afronden van elke opdracht is gevraagd aan de deelnemer om een vragenlijst in te vullen. De resultaten hiervan zijn terug te vinden in de Bijlagen M, N en O. Een samenvatting van de belangrijkste waarnemingen zijn hieronder opgesomd per applicatie.

Levensverzekering (zie Bijlage M)

1. Deelnemer 1 vond de multidimensionale array niet prettig om aan te passen, hoewel de andere deelnemers hiervan geen melding maakten.
2. Deelnemer 2 kampte met problemen in de ontwikkelomgeving, omdat hij vergeten was om te rebuilden. Hierdoor werd tijdens het debuggen de “oude” assemblies gebruikt, hetgeen vreemd gedrag veroorzaakte.
3. Deelnemer 4 had een interpretatiefout van de opdrachtomschrijving, omdat hij de toeslag anders opvatte.
4. Deelnemer 5 had een uitschieter bij de eerste applicatie. Hij had de structuur van de if-statements aangepast en dit later over het hoofd gezien.
5. Deelnemers 6, 7 en 8 uit groep B benoemen expliciet de grote hoeveelheid if-statements in versie A van applicatie Levensverzekering, maar vertellen hierbij ook dat het geen invloed heeft gehad op de analyse of aanpassing.
6. Zowel Deelnemer 6 als Deelnemer 7 zien liever een switch-statement dan if-statements.

Hoekberekening(zie Bijlage N)

7. Deelnemers 1, 2 en 3 geven aan dat de klasse Veelhoek onlogisch / niet netjes is.
8. Deelnemer 1 geeft aan dat hij iets vertraging heeft opgelopen vanwege het verkeerd lezen van de beschrijving.
9. Deelnemer 4 uit groep A heeft iets vertraging opgelopen vanwege een “copy-paste”-fout, waardoor hij verkeerde argumenten meegaf aan een methode uit de klasse WiskundeUtil. Deelnemer 7 uit groep B had een soortgelijke fout gemaakt.
10. Geen van de deelnemers uit groep B had onlogische structuren ontdekt. Deelnemer 8 spreekt zelfs dat door de logische structuur de aanpassing snel gemaakt kon worden.

Belasting(zie Bijlage O)

11. Deelnemer 1 geeft aan dat de klasse TrekkerOplegger lange methoden heeft. Voor de leesbaarheid zou hij gebruik willen maken van hulpmethoden.
12. Zowel Deelnemer 2 als Deelnemer 3 uit groep A geven aan dat alles in een enkele klasse TrekkerOplegger niet logisch/netjes is
13. Alle deelnemers uit groep A hadden direct de oplossing gevonden.
14. Deelnemer 7 had iets vertraging omdat hij eerst anders had gecodeerd dan zoals in de opdrachtomschrijving stond beschreven.
15. Deelnemer 8 geeft aan dat het wel even duurde voordat hij de juiste functies gevonden had.

9. Experiment: conclusies en reflectie

In dit hoofdstuk wordt betekenis gegeven aan de resultaten uit het vorige hoofdstuk. Bovendien wordt per applicatie terugkoppeling gegeven op de gestelde hypotheses.

Conclusies

Per groep

Groep A

De lijnen in Figuur 18 lopen door elkaar, dat erop wijst dat niet telkens een bepaald persoon veel sneller of langzamer was. Zo is bijvoorbeeld te zien dat degene die het langst over de eerste opdracht deed, het snelst was bij de laatste opdracht. Degene die het snelst bij de eerste opdracht was, deed weer het langst over de laatste opdracht. Deelnemers 2 en 4 scoorden beduidend minder bij de applicatie Levensverzekering, waarvan de oorzaak is genoemd in punten 2 en 3 op blz. 59.

Groep B

In Figuur 19 is te zien dat Deelnemer 6 voor elke opdracht het langst bezig was. Dit kan erop wijzen dat deze persoon een verkeerde dag heeft gehad of over mindere capaciteiten beschikt. Echter kunnen natuurlijk zulke conclusies niet gemaakt worden op basis van een drietal metingen, waarvan de eerste meting ook is uitgeschoten door een aanvankelijk verkeerde aanpassing in de if-statements (zie punt 4 op blz. 59). De andere deelnemers zitten dichtbij elkaar, waarbij Deelnemer 8 iets naar boven afwijkt.

Per applicatie

Tegenvoorbeeld: Levensverzekering (Weighted Methods Per Class)

Voor de applicatie Levensverzekering zijn drie flinke uitschieters waar te nemen vanwege verschillende redenen. Deelnemer 2 had vertraging opgelopen vanwege problemen met de buildomgeving (zie punt 2). Dit is dus niet aan de code gerelateerd, waardoor deze meting verder buiten beschouwing wordt gelaten.

Deelnemer 4 had de opdracht verkeerd geïnterpreteerd, waardoor de tests telkens niet slaagden (punt 3 op blz. 59). Deze meting wordt ook niet meegenomen, omdat deze meting niet zo hoog is vanwege de codestructuur.

Deelnemer 5 had op een verkeerde plek in de if-statements een wijziging gemaakt, dat hem later veel tijd kostte om te corrigeren. De opdracht zelf betrof adaptief onderhoud, maar door het introduceren van een nieuwe fout veranderde dit in correctief onderhoud. Deze meting wordt wel meegenomen, omdat deze aan de structuur van de code te relateren is.

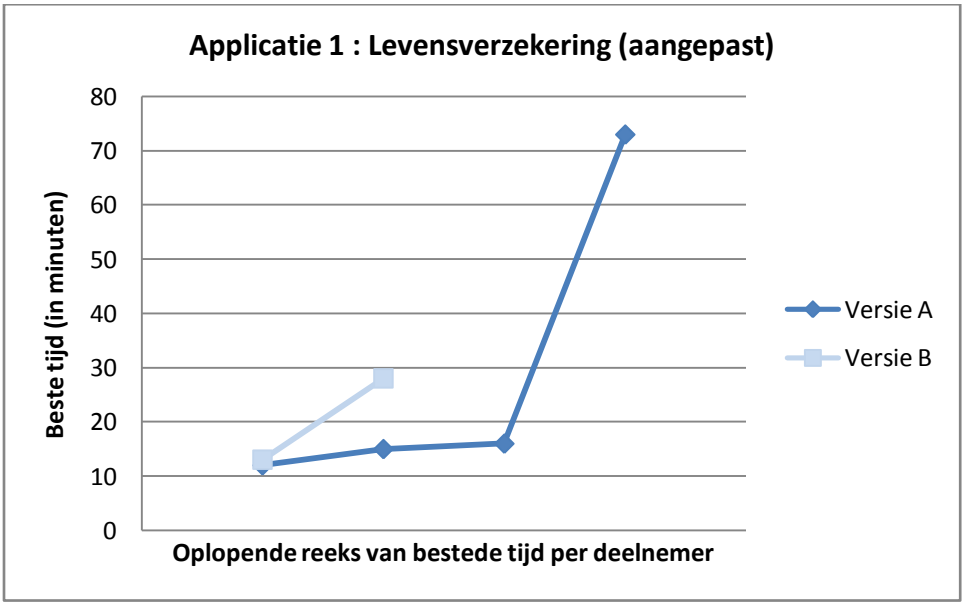
Bij adaptief onderhoud lijken de vele if-statements minder invloed te hebben dan bij correctief onderhoud, omdat alleen gezocht hoeft te worden naar de geschikte plek om de aanpassing te maken. Dit blijkt ook uit het feit dat drie van de vier deelnemers in de vragenlijst aangaf dat er teveel if-

statements aanwezig zijn in de code, maar dat dit geen invloed heeft gehad heeft op de aanpassing (zie punt 5 op blz. 59).

Bij correctief onderhoud hebben de vele if-statements tot gevolg dat er in een van de vele beslissingspaden zich een fout kan bevinden. In dat geval moeten deze paden allemaal doorlopen worden, waardoor de bestede tijd toeneemt.

Van versie B worden dus twee metingen buiten beschouwing gelaten. Dit betekent dat versie A vier metingen heeft, terwijl versie B er enkel twee heeft. Vanwege dit verschil is het niet zinvol om de bestede tijd verder te gaan vergelijken.

De hypothese *"In tegenstelling tot wat de meetwaarden van de metriek WMC suggereren, lijkt het uitvoeren van de opdracht op versie A toch minder tijd te kosten dan op versie B"* kan zowel niet verworpen als niet aangenomen worden.



Figuur 26 Oplopende bestede tijd per versie voor applicatie 1: Levensverzekering (aangepast)

Voorbeeld: Hoekberekening (LCOM)

In Figuur 22 is te zien dat de metingen van de bestede tijd voor beide versies door elkaar heen lopen. Bovendien liggen de lijnen ook niet ver van elkaar verwijderd. De kleine uitschieter van versie B is ook uitgevoerd door Deelnemer 6, welke dus op alle opdrachten langzamer was in zijn groep. Op basis van deze resultaten is er dus geen opmerkelijk verschil in bestede tijd waar te nemen tussen beide versies.

De mediaan is exact gelijk voor beide versies, waarbij de standaarddeviatie en gemiddelde wel iets hoger ligt bij versie B dan bij versie A.

De hypothese *" In overeenstemming tot wat de meetwaarden van de metriek LCOM suggereren, lijkt het uitvoeren van de opdracht op versie A meer tijd te kosten dan op versie B."* kan verworpen worden. Het verschil in de metriek blijkt op basis van deze resultaten geen invloed te hebben op onderhoudbaarheid.

Opmerking

Uit de persoonlijke reacties bleek wel dat versie A met klasse Veelhoek als onlogisch werd beschouwd. Maar liefst drie van de vier deelnemers, die versie A toegewezen had gekregen, lieten dit blijken in de vragenlijst. Echter wordt in deze scriptie de onderhoudbaarheid enkel als bestede tijd gezien en wordt daarom niet meegenomen bij het toetsen van de hypothese.

IJking: Belasting (Bandi-metrieken)

In Figuur 24 liggen alle waarden van de oplopende reeks van versie A boven die van versie B. Er kan echter ook gesteld worden dat twee metingen van versie B boven die van versie A liggen en omgekeerd: twee metingen van versie B liggen boven die van versie A.

Echter geldt voor zowel wel het gemiddelde als de mediaan dat deze hoger liggen voor versie A dan voor versie B, dat er dus op wijst dat versie B beter onderhoudbaar is dan versie A. Bovendien is er ook in de grafiek een trend te ontdekken dat de bestede tijd voor versie A hoger ligt dan versie B. Om deze redenen wordt de hypothese "*Net zoals de uitkomst van het oorspronkelijke experiment, kost het meer tijd om de opdracht uit te voeren op versie B dan op versie A*" aangenomen.

	Bandi	Replicatie
Belasting (Versie A)	36	4
Belasting (Versie B)	57	4

Tabel 42 Aantal deelnemers

	Bandi	Replicatie
Belasting (Versie A)	115	25
Belasting (Versie B)	84	18

Tabel 43 Gemiddeld bestede tijd in minuten

In Tabel 42 staan het aantal deelnemers van het experiment van Bandi ten opzichte van de replicatie. Te zien valt dat bij het oorspronkelijke experiment bijna tien keer zoveel deelnemers bij betrokken waren.

Tabel 43 laat de gemiddelde bestede tijd van het oorspronkelijke experiment en de replicatie voor dit onderzoek zien. Het is opmerkelijk dat bij de replicatie de opdracht gemiddeld ruim vier keer sneller is uitgevoerd. Mogelijke oorzaken hiervan kunnen zijn:

- De capaciteiten van de deelnemers van de replicatie lagen hoger. Hoewel het bij beide experimenten om studenten ging, moesten de deelnemers van de replicatie een toelatingsprocedure doorlopen, waarvan het merendeel is afgewezen.
- De deelnemers bij de replicatie hadden de beschikking over Visual Studio 2005, dat op moment van schrijven de recentste IDE (Integrated Development Environment) van Microsoft is . Tijdens het oorspronkelijke experiment, dat uitgevoerd was in 1998, werd vermoedelijk gebruikt gemaakt van een console met een simpele tekstverwerker.
- In C++ moet rekening gehouden worden met het beheren van objecten in het geheugen, terwijl in C# dit automatisch wordt uitgevoerd door de "garbage collector". In C++ heeft de programmeur dus meer verantwoordelijkheden, dat extra complexiteit met zich meebrengt.

Reflectie

Het experiment is uitgevoerd in een gecontroleerde omgeving met gelijke omstandigheden. Hierdoor kan gecontroleerd worden dat de tijdsmeting correct verloopt, waardoor de validiteit versterkt wordt. Er zijn echter ook punten die een gevaar vormen voor de validiteit van het experiment. Tijdens de reflectie wordt de "construct"-validiteit en de interne validiteit toegelicht. In het volgende hoofdstuk wordt de externe validiteit besproken, omdat dan wordt gekeken wat het experiment betekend heeft voor de "echte" wereld, waarin Info Support zich bevindt in de dagelijkse praktijk.

"Construct"-validiteit

De "construct"-validiteit betekent dat de onafhankelijke en afhankelijke variabelen accuraat de (abstracte) hypothese beslaan. (Perry, et al., 2000).

Na het afronden van een applicatie is er sprake van een "carryover effect" (Kitchenham, et al., 2002). Hiermee wordt het leereffect bedoeld, waardoor de volgende opdracht makkelijker te maken is. Denk aan het wennen van hoe de opdrachten geformuleerd en opgezet zijn. Echter worden de resultaten van de applicaties niet met elkaar vergeleken en elke deelnemer heeft in dezelfde volgorde de opdrachten uitgevoerd.

De bedoeling is om de onderhoudbaarheid van de code te onderzoeken, waarbij uiteraard de kwaliteit van de opdrachtomschrijving zelf buiten de scope valt. De gemeten tijd tijdens het experiment was echter inclusief het lezen van de opdrachtomschrijving. Achteraf gezien was het dus beter om pas te beginnen met de tijdsmeting, als er werd begonnen met het analyseren van de code.

Om deze fout nog enigszins te "corrigeren" is er een leesproef uitgevoerd, zodat een indicatie gegeven kan worden over de benodigde tijd om de opdracht initieel te lezen. Er is dus sprake van een ondergrens, want de benodigde leestijd tijdens het onderhouden van de code is niet inbegrepen. In Bijlage P staat de uitgedeelde instructie voor de leesproef en in onderstaande tabel staan de resultaten.

	Levensverzekering	Hoekberekening	Belasting
Lezer A	5	6	3
Lezer B	7	6	4
Lezer C	8	8	4
Lezer D	10	8	4
Gemiddelde leestijd	**8**	**7**	**4**
Percentage t.o.v. gem. tijd Versie A	**28% (29)**	**27% (26)**	**16% (25)**
Percentage t.o.v. gem. tijd Versie B	**19% (42)**	**23% (31)**	**22% (18)**

Tabel 44 Bestede tijd in minuten om de opdrachtomschrijvingen te lezen

De conclusie die uit Tabel 44 getrokken kan worden is dat de initiële leestijd ongeveer een kwart van de totale bestede tijd in beslag neemt.

In Hoofdstuk 6 is al reeds benoemd dat de aanpassingen tot en met klasseniveau gaan, dat deze alleen adaptief zijn en dat het aspect testbaarheid verwaarloosd is, zodat onderhoudbaarheid niet helemaal gedekt is.

Tot slot kan gesteld worden dat de afgeronde opdrachten enkel op correctheid gecontroleerd zijn, door te kijken of de automatische tests slaagden. Er is echter buiten beschouwing gelaten of de aanpassing zelf ook weer invloed heeft gehad op de onderhoudbaarheid van de applicatie.

Interne validiteit

De interne validiteit betekent dat veranderingen in de afhankelijke variabele veilig kunnen worden toegeschreven aan veranderingen in de onafhankelijke variabelen. (Perry, et al., 2000).

Tijdens het opzetten van de applicaties bleek dat het praktisch onmogelijk is om een enkele metriek te isoleren, zodat andere metrieken gelijk blijven. Hierdoor kan dus een verschil in tijd niet herleid worden naar een specifieke metriek.

Verder is er aangenomen dat alle deelnemers dezelfde capaciteiten hebben. Dit hoeft echter niet zo te zijn, zodat hierdoor ook de bestede tijd beïnvloed kan zijn.

10. Terugkoppeling naar metrieken en enkele meetwaarde

In de voorgaande hoofdstukken 6 t/m 9 staat het experiment beschreven, waarbij drie specifiekere hypotheses zijn opgesteld en getoetst. In dit hoofdstuk wordt de betekenis van het experiment voor metrieken in het algemeen beschreven door nu een uitspraak te doen over de genoemde hypothese uit hoofdstuk 4:

Als in de wetenschappelijke literatuur een metriek gepositioneerd wordt als een indicator voor onderhoudbaarheid, dan lijkt dit op basis van eigen interpretatie niet altijd te kloppen. De verwachting is dat de beweringen uit de eigen interpretatie ook empirisch aangetoond kunnen worden.

Tot slot wordt ook kort besproken wat het experiment heeft betekend voor de enkele meetwaarde, waarnaar Info Support op zoek is.

Metrieken

De hypothese uit hoofdstuk 4 kan op basis van het experiment niet bevestigd worden.

Het tegenvoorbeeld uit het experiment is niet bevestigd of afgewezen, vanwege problemen die niet aan de code gerelateerd waren.

Het voorbeeld, waarin de verwachting was dat de bestede tijd daadwerkelijk verschilden, is afgewezen, omdat er geen verschil was in de bestede tijd. Dit bleek uit de resultaten van de applicatie Hoekberekening.

Dit geeft een signaal af dat een verschuiving van de meetwaarden niet een verandering van onderhoudbaarheid tot gevolg hoeft te hebben.

De replicatie gaf eenzelfde trend weer als het oorspronkelijke experiment, zodat met meer zekerheid gesteld kan worden dat de eigen omgeving correct is. Het bleek echter dat de metrieken niet goed geïsoleerd waren, zodat over de Bandi-metrieken verder geen uitspraak gedaan kan worden.

Externe validiteit

De externe validiteit betekent in hoeverre de resultaten te generaliseren zijn buiten het experiment (Perry, et al., 2000).

Het aantal van acht deelnemers, waarvan vier per versie, is te beperkt om algemeen geldende conclusies te trekken. Ook bestond de populatie uit bijna afgestudeerde studenten. Het gevaar hierin zit in hoeverre dit te generaliseren kan worden naar de algemene groep softwareontwikkelaars.

Het experiment is uitgevoerd op kleine applicaties. Dit was vanwege de beperkte tijd die beschikbaar was voor het experiment. In de dagelijkse praktijk heeft Info Support te maken met applicaties die groter zijn qua omvang. Het is de vraag of de conclusies in een “kleine” omgeving doorgetrokken kunnen worden naar een “grote” omgeving.

De gebruikte programmeertaal voor het experiment was C#. De aanname is dat de resultaten schaalbaar zijn naar andere objectgeoriënteerde programmeertalen, omdat ze dezelfde concepten kennen als overerving en polymorfisme. Het verschil tussen objectgeoriënteerde talen zit voornamelijk in de syntax en niet in de semantiek. De beschikbare klassenbibliotheken zijn per objectgeoriënteerde programmeertaal verschillend, maar omdat In het experiment hiervan geen gebruik is gemaakt, valt dit te verwaarlozen.

In het experiment worden specifieke metrieken in een specifieke situatie onderzocht. Het kan dus betekenen dat “toevallig” een ongeschikte metriek is gekozen. Hoewel de aanname is gemaakt dat de onderhoudstaak representatief was voor een onderhoudstaak, kan het zo zijn dat andere opdrachten (met gelijke meetwaarden) verschillende resultaten geven.

Dit onderzoek heeft slechts een selectie van alle bestaande metrieken belicht. Het experiment heeft vervolgens hiervan weer een subset genomen. Het is niet mogelijk om de bevindingen van een bepaalde metriek te generaliseren voor alle metrieken. Het lijkt echter wel nuttig om het eerdergenoemde signaal, dat verschuiving van meetwaarde geen verschil hoeft te maken voor onderhoudbaarheid, mee te nemen bij het interpreteren van verschillen in meetwaarden van andere metrieken. Blindelings vertrouwen op meetwaarden, zonder de semantiek mee te nemen, brengt risico op misinterpretatie met zich mee.

Enkele meetwaarde voor onderhoudbaarheid

Om tot een enkele meetwaarde voor onderhoudbaarheid te komen, moet er een voldoende historische basis aanwezig zijn, zodat er een verband bepaald kan worden tussen de metrieken en de onderhoudbaarheid. Het experiment heeft echter een enkele metingen gedaan per metriek.

Het aantonen van het tegenvoorbeeld is niet gelukt vanwege externe problemen. Als dit wel gelukt was, dan had dat een aanwijzing kunnen geven dat een enkele meetwaarde onmogelijk is, omdat een verschuiving van een meetwaarde ook juist het tegenovergestelde kan betekenen voor onderhoudbaarheid.

De uitkomst van de applicatie Hoekberekening verzwakt wel de haalbaarheid van een enkele meetwaarde, omdat tegen de verwachting in de metriek geen invloed had op onderhoudbaarheid.

11. Conclusies, aanbevelingen en toekomstig werk

Conclusies

Het begrip onderhoudbaarheid kent verschillende definities, waardoor het onduidelijk is wat er precies onder verstaan wordt. In deze scriptie wordt onderhoudbaarheid opgesplitst in de drie aspecten analyseerbaarheid, aanpasbaarheid en testbaarheid uit het onderhoudsproces. Daarnaast is er een nieuwe definitie gedefinieerd, waarin onderhoudbaarheid als “de bestede tijd per taakcomplexiteit per geschiktheid programmeur” wordt beschouwd.

Er zijn meerdere onderzoeken uitgevoerd om de relatie tussen metrieken en onderhoudbaarheid aan te tonen. Hierin is echter verschillend naar onderhoudbaarheid gekeken.

Uit eigen analyse blijkt dat als de meetwaarde van een metriek toeneemt, dan kan zowel de onderhoudbaarheid toe- of afnemen. Dit komt omdat de semantiek van de code niet naar voren komt in de meetwaarde van een metriek. Opmerkelijk is dat bestaande onderzoeken maar een enkele kant hiervan toelichten en niet openlijk relevante tegenvoorbeelden noemen. Misschien zijn sommige onderzoekers “gebiassed”, omdat ze op zoek zijn naar een onderzoeksresultaat waarbij de metriek een relatie met onderhoudbaarheid moet hebben.

Om tot een enkele meetwaarde voor onderhoudbaarheid te komen, is er allereerst gekeken naar de twee soortgelijke benaderingen Static Confidence Factor en Maintainability Index. Bij beiden wordt een kwaliteitsattribuut vertaald naar een enkele waarde. Het was echter niet duidelijk waarin deze meetwaarden uitgedrukt kon worden, zodat het kwaliteitsattribuut zelf dus niet gemeten wordt.

Het meten van onderhoudbaarheid is een meting voor voorspelling, want het gaat om de verwachte bestede tijd. De twee andere dimensies taakcomplexiteit en geschiktheid van programmeur zijn lastig te meten, omdat ze afhankelijk zijn van meerdere factoren. Bovendien zijn deze niet op basis van codeanalyse te verkrijgen.

Door het uitvoeren van een experiment is het mogelijk om deze dimensies bij benadering constant te houden. De bestede tijd kan heeft daardoor een direct verband met de onderhoudbaarheid.

Het experiment heeft een signaal afgegeven dat een verschuiving van de metrieken geen invloed hoeft te hebben op onderhoudbaarheid en dat daarom voorzichtig moet worden omgegaan met codemetrieken. Metrieken confronteren je met feiten over de code, zodat je erover gaat nadenken. Het zijn dus signalen, maar moeten dus wel gecontroleerd worden door een expert.

Aanbevelingen voor Info Support

Het experiment is uitgevoerd in een laboratorium op kleine applicaties. Om meer kennis op te doen wat metrieken betekenen voor de onderhoudbaarheid van echte applicaties binnen Info Support, lijkt een “business intelligence”- aanpak een goede keuze. Hierin wordt telkens de cirkel van verzamelen, analyseren en toepassen doorlopen (Den Hamer, 2005).

Om meer kennis op te doen wat metrieken betekenen voor de onderhoudbaarheid van echte applicaties, moeten gegevens gestructureerd verzameld worden van meerdere projecten en opgeslagen worden in een data warehouse. Het GQM-paradigma kan hierbij goed helpen om meetdoelen zo expliciet mogelijk te formuleren (Basili, et al., 1984). Om onderhoudbaarheid te meten is er dus meer benodigd dan alleen codemetrieken en de bestede tijd. Ook eigenschappen van de taak en programmeur moeten worden vastgelegd..

Vervolgens moeten de gegevens geanalyseerd worden. Fenton en Neil adviseren om hiervoor causal modeling met Bayesian Belief Nets toe te passen, waardoor het beter mogelijk is om het oorzakelijk verband te achterhalen (Fenton, et al., 2000).

Tot slot moet op basis van de analyse acties ondernomen worden. Dit kan bijvoorbeeld het refactoren van code zijn, waardoor codemetrieken veranderen. Het is essentieel om vervolgens deze gegevens weer te verzamelen, zodat de cirkel opnieuw doorlopen wordt.

Toekomstig wetenschappelijk werk

Een experiment in een laboratorium lijkt de beste keuze om zo goed mogelijk onderhoudbaarheid te meten, zoals in deze scriptie gedefinieerd is. Dit biedt namelijk een gecontroleerde omgeving waarbij zoveel mogelijk factoren constant gehouden kunnen worden.

Het uitgevoerde experiment heeft verschillende beperkingen. Deze kunnen weggenomen door het experiment grootschaliger uit te voeren. De beperkingen en ideale situatie staan weergegeven In Tabel 45.

Beperking	Ideale situatie
Er waren slechts acht deelnemers, die allemaal nog studenten zijn.	Een groter aantal gelijke deelnemers, die al meerdere jaren actief zijn als ontwikkelaar.
Per applicatie waren slechts twee versies.	Meerdere versies waarin telkens de meetwaarde van de metriek oploopt.
De opdrachten bestonden enkel uit adaptief onderhoud.	Naast adaptieve ook correctieve en perfectieve taken definiëren.
Er zijn slechts enkele metrieken onderzocht	Een groter aantal verschillende metrieken.

Tabel 45 Beperkingen van het experiment met bijbehorende ideale situatie

Door nader onderzoek te doen in richting van de ideale situatie kan beter bepaald worden wat de betekenis is van codemetrieken op onderhoudbaarheid.

Bibliografie

Aggarwal K. K. [et al.] Empirical Study of Object-Oriented Metrics [Tijdschrift] // Journal of Object Technology. - [sl] : ETH Zurich, December 2006. - 8 : Vol. 5. - pp. 149-173.

Bandi R., Vaishnavi V. en Turk D. Appendix G: Computation of metrics [Rapport]. - [sl] : IEEE Transactions on Software Engineering, 2002. - http://www.biz.colostate.edu/faculty/dant/pages/papers/IEEETrxSE-OOMetrics/20020917/AppendixG-MetricsComputation.pdf.

Bandi R.K., Vaishnavi V.K. en Turk D.E. Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics [Tijdschrift] // IEEE Trans. Softw. Eng.. - Piscataway, NJ, USA : IEEE Press, 2003. - 1 : Vol. 29. - pp. 77-87. - 0098-5589.

Basili V.R. en Weiss D.M. A Methodology for Collecting Valid Software Engineering Data [Tijdschrift] // IEEE Trans. Software Eng.. - [sl] : IEEE Press, 1984. - 6 : Vol. 10.

Bocco M.G., Moody D.L. en Piattini M. Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation: Research Articles [Tijdschrift] // J. Softw. Maint. Evol.. - New York, NY, USA : John Wiley & Sons, Inc., 2005. - 3 : Vol. 17. - pp. 225-246. - 1532-060X.

Boehm B.W., Brown J.R. en Kaspar J.R. Characteristics of Software Quality (TRW series of software technology) [Boek]. - Amsterdam : American Elsevier, 1978. - 0444851054.

Breesam K.M. Metrics for Object-Oriented Design Focusing on Class Inheritance Metrics [Conferentie] // DEPCOS-RELCOMEX '07: Proceedings of the 2nd International Conference on Dependability of Computer Systems. - Washington, DC, USA : IEEE Computer Society, 2007. - pp. 231-237. - 0769528503.

Briand L.C. en Wüst J. Empirical Studies of Quality Models in Object-Oriented Systems [Tijdschrift] // Advances in Computers. - [sl] : Academic Press, 2002. - Vol. 56. - pp. 98-167.

Cavano J.P. en McCall J.A. A framework for the measurement of software quality [Tijdschrift] // SIGSOFT Softw. Eng. Notes. - New York, NY, USA : ACM Press, 1978. - 5 : Vol. 3. - pp. 133-139. - 0163-5948.

Chae H.S. [et al.] Using Metrics for Estimating Maintainability of Web Applications: An Empirical Study [Conferentie] // 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007). - Melbourne, Australia : IEEE Computer Society, 2007. - pp. 1053-1059. - 0769528414.

Chidamber S. en Kemerer C. A metrics suite for object oriented design [Tijdschrift] // IEEE Transactions on Software Engineering. - Piscataway, NJ, USA : IEEE Press, June 1994. - 6 : Vol. 20. - pp. 476-493. - 0098-5589.

Chidamber S.R. en Kemerer C.F. Towards a metrics suite for object-oriented design [Conferentie] // OOPSLA'91. - 1991. - pp. 197-211.

Coleman D. [et al.] Using Metrics to Evaluate Software System Maintainability [Tijdschrift] // Computer. - Los Alamitos, CA, USA : IEEE Computer Society Press, 1994. - 8 : Vol. 27. - pp. 44-49. - 0018-9162.

Dagpinar M. en Jahnke J.H. Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison [Conferentie] // WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering. - Washington, DC, USA : IEEE Computer Society, 2003. - p. 155. - 0769520278.

Den Hamer P. De organisatie van business intelligence [Boek]. - Den Haag : Sdu Uitgevers b.v., 2005. - Vol. Inleiding.

Fenton N.E. en M. Neil Software metrics: roadmap [Conferentie] // ICSE '00: Proceedings of the Conference on The Future of Software Engineering. - Limerick, Ireland : ACM Press, 2000. - pp. 357-370. - 1581132530.

Fenton N.E. en Pfleeger S.L. Software Metrics: A Rigourous & Practical Approach [Boek]. - Boston, MA : PWS Publishing Company, 1997. - Second Edition. - 0534954251.

Fowler M. Refactoring: improving the design of existing code [Boek]. - Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. - 0201485672.

Gamma E. [et al.] Design Patterns [Boek]. - [sl] : Addison-Wesley Professional, 1995. - 0201633612.

Garvin D.A. Managing Quality: The Strategic and Competitive Edge [Boek]. - New York : Free Press, 1988. - 0029113806.

Genero M. [et al.] Using Metrics to Predict OO Information Systems Maintainability [Conferentie] // CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering. - London, UK : Springer-Verlag, 2001. - pp. 388-401. - 3540422153.

Halstead M.H. Elements of Software Science (Operating and programming systems series) [Boek]. - New York, NY : Elsevier Science Inc., 1977. - 0444002057.

Harrison R., Counsell S. en Nithi R. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems [Tijdschrift] // The Journal of Systems and Software. - New York, NY, USA : Elsevier Science Inc., 2000. - 2-3 : Vol. 52. - pp. 173-179. - 0164-1212.

Heitlager I, Kuipers T. en Visser J. A Practical Model for Measuring Maintainability [Conferentie] // In proceedings of the 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007). - [sl] : IEEE Computer Society Press, 2007.

Henderson-Sellers B. Object-oriented metrics: measures of complexity [Boek]. - Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1996. - 0132398729.

Hitz M. en Montazeri B. Chidamber & Kemerer's Metrics Suite: A Measurement Theory Perspective [Tijdschrift] // IEEE Transactions on Software Engineering. - 1996. - 4 : Vol. 22. - pp. 276-270. - ISERN-98-07 30.

IEE Std. 610.12-1990 // Glossary of Software Engineering Terminology. - Los Alamitos, Calif. : IEEE CS Press, 1993. - 1048-06T.

IEEE IEEE Std 1219-1998: IEEE Standard for Software Maintenance [Rapport]. - [sl] : IEEE, 1998.

IEEE IEEE Std. 1219-1993: Standard for Software Maintenance [Rapport]. - 1993.

ISO ISO/IEC FIDS 14764 : Software Engineering - Software Maintenance [Rapport]. - 2000.

ISO/IEC JTC 1 DTR Ballot on ISO/IEC 9126: Software Engineering - Product Quality Part 3 - Internal Metrics [Rapport]. - New York, NY : Software Engineering Secretariat : CANADA (SCC), 2001.

Jansen P., Krikhaar R. en Dijkstra F. Towards a Single Software Quality Metric [Online] // TIOBE Software - The Coding Standards Company. - 31 Juli 2007. - http://www.tiobe.com/standards/confidence_factor.htm.

Jones C. Programming Productivity [Boek]. - New York : McGraw-Hill, Inc., 1986. - 0070328110.

Karlsson E Software Reuse - A Holistic Approach [Boek]. - England : JohnWiley & Sons, 1995.

Kitchenham B.A. [et al.] Preliminary guidelines for empirical research in software engineering [Tijdschrift] // IEEE Transactions on Software Engineering . - Piscataway, NJ, USA : IEEE Press, 2002. - 8 : Vol. 28. - pp. 721-734. - 0098-5589.

Lehman M. en Fernandez-Ramil J. Chapter 1: Software Evolution [Sectie van boek] // Software Evolution and Feedback: Theory and Practice. - [sl] : Wiley, 2006.

Li W. en Henry S. Maintenance metrics for the object oriented paradigm [Conferentie] // Proceedings of the First International Software Metrics Symposium. - Baltimore, MD : [sn], 1993. - pp. 52-60.

Li W. en Henry S. Object-oriented metrics that predict maintainability [Tijdschrift] // J. Syst. Softw.. - New York, NY, USA : Elsevier Science Inc., 1993. - 2 : Vol. 23. - pp. 111-122. - 0164-1212.

Littlewood B. Forecasting software reliability [Tijdschrift] // Software Reliability, Modeling and Identification (Lecture Notes in Computer Science). - New York : Springer-Verlag, 1988. - Vol. 341. - pp. 141-209.

LocMetrics LocMetrics - Source Code Line Counting Tool [Online] // LocMetrics. - Februari 2007. - 3 September 2007. - http://www.locmetrics.com/.

McCabe T.J. A complexity measure [Conferentie] // ICSE '76: Proceedings of the 2nd international conference on Software engineering. - San Francisco, California, United States : IEEE Computer Society Press, 1976. - p. 407.

McConnell S. Code Complete [Boek]. - Redmond : Microsoft Press, 2004. - Second Edition. - 0735619670.

NDepend NDepend [Online] // NDepend. - 2007. - 11 Juli 2007. - http://www.ndepend.com.

Perry D.E., Porter A.A. en Votta L. Empirical studies of software engineering: a roadmap [Conferentie] // ICSE '00: Proceedings of the Conference on The Future of Software Engineering. - Limerick, Ireland : ACM Press, 2000. - pp. 345-355. - 1581132530.

Pfleeger S.L. Software Engineering: Theory and Practice [Boek]. - Upper Saddle River, NJ, USA : Prentice Hall PTR, 2001. - 0130290491.

Shi N. en Olsson R.A. Reverse Engineering of Design Patterns from Java Source Code [Conferentie] // ASE '06: Proceedings of the 21st IEEE International Conference on Automated Software Engineering. - Washington, DC, USA : IEEE Computer Society, 2006. - pp. 123-134. - 0-7695-2579-2.

Sommerville I. Software engineering [Boek]. - Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 1995. - Fifth edition. - 0201427656.

Swanson B. E. The dimensions of maintenance [Conferentie] // Intl. Conf. on Software Engineering. - San Francisco, California : IEEE Computer Society, 1976. - pp. 492-497. - 768959.

Van Willegen J.J.H. Extractie van dode code: statische analyse in combinatie met dynamische analyse [Rapport]. - Amsterdam : Universiteit van Amsterdam, 2006.

Wood M. [et al.] Multi-method research: an empirical investigation of object-oriented technology [Tijdschrift] // J. Syst. Softw.. - New York, NY, USA : Elsevier Science Inc., 1999. - 1 : Vol. 48. - pp. 13-26. - 0164-1212.

A. Persoonlijke vragenlijst

- De informatie die vergaard wordt in deze vragenlijst wordt alleen gebruikt in het kader van dit onderzoek.
- Persoonlijke identificatie wordt op geen enkele wijze verstrekt en wordt vertrouwelijk behandeld.

Persoonlijke gegevens

Naam	
Leeftijd	
Adres*	
Postcode*	
Woonplaats*	

Opleiding

Voor welke opleiding ben je nu aan het afstuderen bij Info Support?	
Op welke hogeschool of universiteit volg je deze opleiding?	
Welke titel ontvang je indien het afstuderen succesvol verloopt?	Bachelor / Master

Ervaring programmeertalen

	Vanaf welk jaartal ben je hiermee begonnen?	**Hoeveelheid ervaring heb je omgerekend naar volledige werkweken (40 uur)?**
Niet-objectgeoriënteerde programmeertalen		
Objectgeoriënteerde talen		
.NET programeertalen		

***= Optioneel. Alleen invullen bij interesse voor ontvangen van eindscriptie**

B. Onderschepte problemen en genomen acties na uitvoeren van pilot

Algemeen

Probleem	Ondernomen actie
Windows Updates werden gedownload tijdens de pilot en er kwam melding dat deze geïnstalleerd moesten worden.	Windows Updates is uitgeschakeld op de image.
De beide systemen die de twee proefdeelnemers gebruikten, waren gebaseerd op dezelfde Virtual PC image. Ze hadden dezelfde computernaam, hetgeen een conflict veroorzaakte in het netwerk.	Tijdens het klaarzetten van de opstelling voor het echte experiment in het laboratorium zijn alle computernamen uniek gemaakt.
De vraaggesprekken met de deelnemers leverde informatie op, dat ook interessant is om te weten voor echte experiment. Deze wordt echter niet gestructureerd vastgelegd.	Na het afronden van elke applicatie vult de deelnemer een vragenlijst in. De structuur van deze vragenlijsten kan teruggevonden worden in Bijlagen M t/m O.

Tabel 46 Algemene problemen tijdens proefdraaien met applicaties en ondernomen acties voor echte experiment

Levensverzekering

Probleem	Ondernomen actie
De opdrachtomschrijving bevatte enkele spelfouten, waardoor de deelnemers onnodig afgeleid werden.	Deze spelfouten zijn gecorrigeerd in de opdrachtomschrijving.
In Tabel 51 stond niet vermeld dat de gemiddelde looptijd in jaren worden uitgedrukt.	Er wordt nu wel vermeld dat de looptijd in jaren is.
Bij de uitleg over de premie per maand stond een “verdwaalde” zin die niet meer in de context paste.	Deze zin is verwijderd uit de opdrachtomschrijving
Eerst werd er een figuur getoond, terwijl dit later pas in de tekst benoemd werd. Andersom zou prettiger lezen zijn.	Eerst wordt figuur in tekst genoemd en dan pas weergegeven.
Uit de opdrachtformulering werd niet genoeg duidelijk dat de “nieuwe” sterftekans uit Tabel 52 gehaald moest worden. Ook had dit beter naar voren moeten komen tijdens het bespreken van de tabel.	Dit is duidelijker verwoord in de opdrachtomschrijving. Bovendien is deze opmerking ook extra benoemd tijdens experiment.
De eerste test (=huidige functionaliteit) slaagde en de tweede (=nieuwe functionaliteit) faalde. Dit is ook de bedoeling, maar dit zorgde toch voor onzekerheid bij een van de proefdeelnemers.	Door expliciet te noemen in de opdrachtformulering, wordt er meer duidelijkheid geschept.
De ontwikkelomgeving Visual Studio 2005 stond ingesteld op de standaard instellingen, terwijl de proefdeelnemers gewend waren aan de C# instellingen.	In de sheets (zie Bijlage B) wordt benoemd dat eventuele voorkeursinstellingen voor de tijd ingesteld kunnen worden.

Tabel 47 Problemen tijdens proefdraaien met applicatie Levensverzekering en ondernomen acties voor echte experiment

Hoekberekening

Probleem	Ondernomen actie
In de uitleg van de cosinusregel werd verteld dat zijde AB als letter b beschouwd moest worden in de cosinusregel, terwijl dit letter c moest zijn.	Dit is aangepast in de opdrachtomschrijving.

Tabel 48 Problemen tijdens proefdraaien met applicatie Hoekberekening en ondernomen acties voor echte experiment

Belasting

Probleem	Ondernomen actie
In verandering 1 wordt een nieuwe belasting voor een oplegger gedefinieerd. Het is onduidelijk of in verandering 2 (voor het berekenen van de totale belasting) uitgegaan moet worden van de oude of nieuwe belasting.	De belasting van de oplegger wordt expliciet “Nieuwe belasting oplegger” genoemd en hiernaar wordt letterlijk verwezen bij verandering 2.

Tabel 49 Problemen tijdens proefdraaien met applicatie Belasting en ondernomen actie voor echte experiment

C. Sheets van presentatie voorafgaand aan experiment

EXPERIMENT (CHRIS WOOLDERINK)

19 juni 2007

Agenda

- 9:00 - 9:30 Ontvangst
- 9:30 – 9:45 Presentatie
- 9:45 – 10:55 Experiment (start)
- 10:55 – 11:00 Verloting boek
- 11:00 – 11:30 Lunch in de kantine
- 11:30 – 13:00 Experiment (vervolg)

Waarom dit onderzoek?

- Relatie tussen onderhoudbaarheid van code en codemetrieken
- Onderhoudbaarheid is uitgedrukt in bestede tijd
  - Tijdregistratie is essentieel
- Verschillende versies van de voorbeeldprogramma's

Opbouw experiment

- Persoonlijke vragenlijst
  - Informatie wordt alleen gebruikt voor onderzoek
  - Persoonlijke identificatie wordt niet vrijgegeven
  - Adres invullen indien je eindscriptie thuis wilt ontvangen
- Drie voorbeeldapplicaties (C# 2.0, Visual Studio 2005)
  - Levensverzekering
  - Hoekberekening
  - Belasting
- Opdrachten worden apart uitgedeeld; bij afronden opdracht wordt volgende opdracht uitgereikt

Instructies (1)

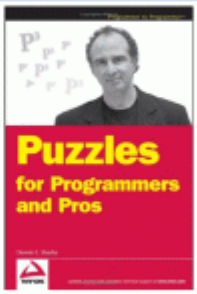
- Gebruik van internet voor raadplegen informatiebronnen is toegestaan
- Overleg tussen deelnemers is niet toegestaan
  - Tijdens lunch niet over experiment praten
- Stel gerust vragen bij onduidelijkheden
  - Er wordt geen antwoord gegeven bij onvoldoende technische kennis
  - Indien ik antwoord geef, dan deel ik dit ook mee aan de andere deelnemers.

Instructies (2)

- **Tijdregistratie!**
  - Tijd start bij uitreiken van opdracht
  - Tijd eindigt bij slagen van test(s)
- Voorkeursinstellingen Visual Studio 2005?

Bedankt!

- Als dank voor jouw deelname krijg je straks de eindscriptie thuisbezorgd (indien je adres invult op persoonlijke vragenlijst)
- Als dank voor jouw deelname maak je kans op een boek(je) →



D. Opdrachtomschrijving applicatie 1 : Levensverzekering

Schrijf hier begintijd hier op wanneer je begint met lezen:

Inleiding

Levensverzekering

Door het afsluiten van een levensverzekering kan men zich verzekeren tegen het plotseling overlijden van een persoon. Indien dit gebeurt, dan wordt het verzekerde bedrag (zoals opgenomen in de afgesloten verzekering) uitgekeerd aan de nabestaanden. Als tegenprestatie draagt de persoon gedurende de looptijd van de verzekering elke maand een premie af aan de verzekeraar.

Berekening van premie per maand

De verzekeraar wil op een eenduidige manier de berekening van de premie per maand beschikbaar stellen voor toekomstige .NET 2.0 applicaties binnen het bedrijf. Er is daarom een C# class library ontwikkeld, waarmee op basis van eigenschappen van de persoon en gewenste verzekering de premie wordt teruggegeven.

Sterftekans

De sterftekans is een invloedrijke factor op het berekenen van de premie per maand. Hoe hoger de sterftekans, des te meer risico loopt de verzekeraar om het verzekerde bedrag vroegtijdig uit te keren. In de huidige versie van de class library wordt de sterftekans bepaald aan de hand van twee eigenschappen:

1. De leeftijdsklasse van de persoon.
2. De looptijdklasse van de gewenste verzekering.

De mogelijke combinaties van deze klassen en bijbehorende sterftekans is weergegeven in Tabel 50. Er is een opdeling in klassen gemaakt om het voorbeeld eenvoudig te houden.

	Junior leeftijd (tussen 20 en 35 jaar)	**Medior leeftijd** (tussen 35 en 50 jaar)	**Senior leeftijd** (tussen 50 en 65 jaar)
Korte looptijd (tussen 0 en 15 jaar)	5%	23%	40%
Gemiddelde looptijd (tussen 15 en 30 jaar)	30%	47%	81%
Lang looptijd (tussen 30 en 45 jaar)	55%	71%	98%

Tabel 50 Sterftekans op basis van leeftijd en looptijd

Verzekerde bedrag

Het verzekerde bedrag wordt uitgekeerd aan de nabestaanden, indien de persoon komt te overlijden. Dit staat opgenomen in de verzekering.

Totaalbedrag "zonder winst"

Het totaalbedrag is het bedrag dat de persoon in totaal moet afdragen tot en met het einde van de looptijd. Dit bedrag is afhankelijk van de sterftekans (zie Tabel 50) en het verzekerde bedrag. Ook is het afhankelijk van het gemiddelde percentage tijd dat volbracht is van de looptijd bij overlijden. Tijdens deze periode heeft de persoon namelijk wel premie afgedragen. Hiervoor wordt een gemiddelde van 50% van de looptijd genomen.

$$Totaalbedrag\ zonder\ winst\ = \frac{sterftekans\ \times verzekerde\ bedrag}{1 - (1 - 0.50) \times sterftekans}$$

Totaalbedrag "met winst"

Door het totaalbedrag zonder winst te vermenigvuldigen met de winstfactor, wordt het totaalbedrag met winst verkregen. De winstfactor in dit voorbeeld is vastgesteld op 2.0.

$$Totaalbedrag\ met\ winst = Totaalbedrag\ zonder\ winst \times 2.0$$

Premie per maand

Het totaalbedrag moet worden uitgesmeerd over het aantal maanden dat de verzekering loopt. Aangezien er dus verschillende klassen worden gebruikt voor de looptijd, wordt in dit voorbeeld het gemiddelde genomen van de betreffende klasse, zoals weergegeven in Tabel 51.

Klasse looptijd	Gemiddelde looptijd in jaren
Korte looptijd (tussen 0 en 15 jaar)	8
Gemiddelde looptijd (tussen 15 en 30 jaar)	23
Lang looptijd(tussen 30 en 45 jaar)	38

Tabel 51 De gehanteerde gemiddelde looptijd per klasse

Op basis hiervan wordt het aantal lopende maanden bepaald:

$$Aantal\ lopende\ maanden = Gemiddelde\ looptijd\ \times 12$$

De premie per maand kan vervolgens als volgt berekend worden:

$$Premie\ per\ maand = \frac{Totaalbedrag\ met\ winst}{Aantal\ lopende\ maanden}$$

Korting voor bestaande klant

Indien de persoon een bestaande klant is van de verzekeraar, dan wordt het bedrag vermenigvuldigd met een kortingsfactor. In dit voorbeeld is deze kortingsfactor vastgesteld op 0.96.

$$Premie\ per\ maand = Premie\ per\ maand \times 0.96$$

Probleemstelling

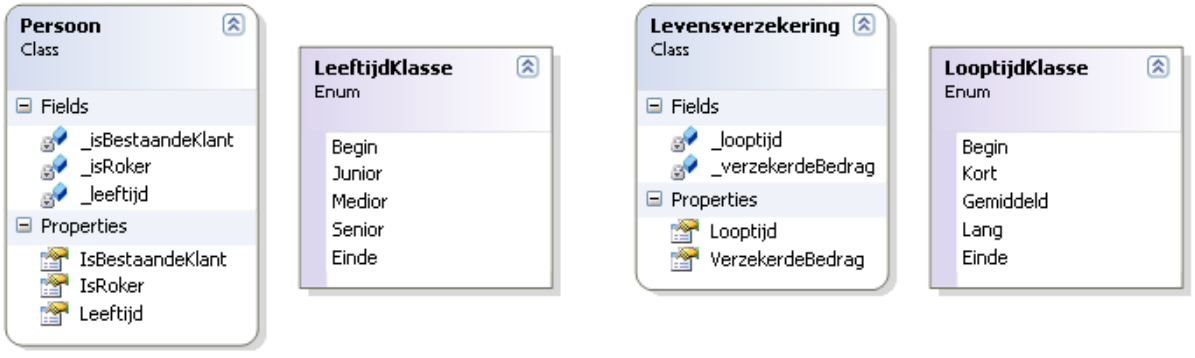
Er is een nieuwe tabel beschikbaar gekomen, waarin een verhoogde sterftekans is opgenomen voor het geval dat iemand rookt. **De percentages voor niet-rokers zijn gelijk gebleven met de vorige tabel**.

	Junior leeftijd (tussen 20 en 35 jaar)		**Medior leeftijd** (tussen 35 en 50 jaar)		**Senior leeftijd** (tussen 50 en 65 jaar)	
Korte looptijd (tussen 0 en 15 jaar)	**Niet** 5%	**Wel** 7%	**Niet** 23%	**Wel** 29%	**Niet** 40%	**Wel** 44%
Gemiddelde looptijd (tussen 15 en 30 jaar)	**Niet** 30%	**Wel** 42%	**Niet** 47%	**Wel** 59%	**Niet** 81%	**Wel** 89%
Lang looptijd (tussen 30 en 45 jaar)	**Niet** 55%	**Wel** 77%	**Niet** 71%	**Wel** 89%	**Niet** 98%	**Wel** 99%

Tabel 52 Sterftekans op basis van leeftijd, looptijd en wel/niet roker zijn

Bovenop deze nieuwe percentages heeft de verzekeraar besloten om het roken ook nog extra te ontmoedigen door een rokerstoeslag in te voeren. Deze toeslag wordt berekend op het totaalbedrag “zonder winst”. De rokerstoeslag wordt berekend aan de hand van een rokersfactor met de waarde 1.1.

Klassen, enums en interfaces



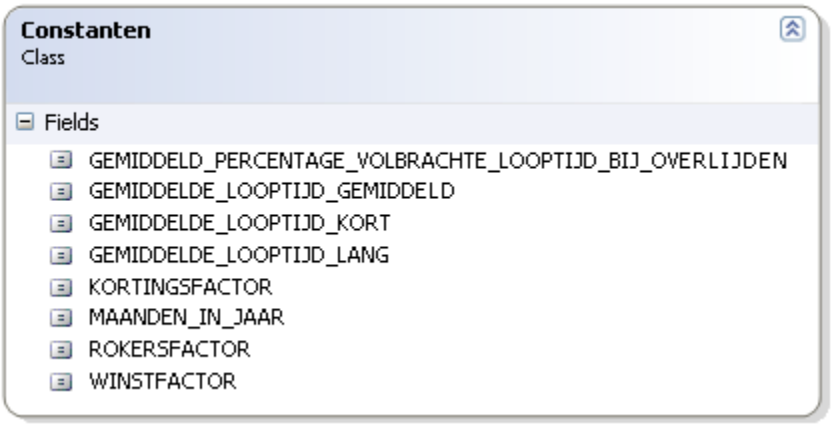
Figuur 27 De domeinobjecten in de applicatie

In Figuur 27 zijn de domeinobjecten weergegeven. Voor de klasse Persoon is al de property opgenomen of deze wel of niet rookt. Daarnaast kan worden opgevraagd of deze persoon een bestaande klant is en

welke leeftijd deze persoon heeft, welke van het type enum LeeftijdKlasse is, die daarnaast is weergegeven.

Voor een Levensverzekering wordt het verzekerde bedrag bijgehouden. De looptijd is ook als property beschikbaar van het type enum LooptijdKlasse, welke daarnaast is weergegeven.

In Figuur 28 staat de klasse Constanten weergegeven, welke de vaste waarden bevat die al eerder benoemd zijn.



Figuur 28 Een aparte klasse voor het bijhouden van constante waarden

In Figuur 29 staat de interface beschreven welke een calculator moet implementeren. Deze interface bevat slechts een enkele methode, welke op basis van een persoon en een gewenste verzekering een premie per maand teruggeeft.

Figuur 29 Interfacedefinitie voor de calculator

Ook is er een klasse Calculator opgenomen in het programma, maar deze is voor het belang van dit experiment niet opgenomen in deze beschrijving.

Opdrachtformulering

Open de solution “Levensverzekering”. De volgende twee deelopdrachten moeten worden uitgevoerd:

1. Pas de broncode aan zodat voor het berekenen van de maandpremie de “nieuwe” sterftekans voor een roker (zoals is weergegeven in de Tabel 52) wordt gebruikt in de berekening. (Opmerking: Zoals eerder vermeld is de property IsRoker al opgenomen in de klasse Persoon).
2. Pas de broncode aan zodat tijdens het berekenen van de maandpremie een rokerstoeslag wordt verrekend op het totaalbedrag “zonder winst”. (Opmerking: de rokersfactor is al reeds opgenomen in de klasse Constanten).

Om te controleren of beide opdrachten correct zijn uitgevoerd, moeten de volgende tests slagen:

1. berekenPremiePerMaandHuidigeFunctionaliteitTest()
2. berekenPremiePerMaandNieuweFunctionaliteitTest()

Deze zijn al gedefinieerd in het Test Project “LevensverzekeringTest”. In de aangeleverde broncode slaagt de eerste test wel (huidige functionaliteit), maar de tweede test niet (met de nog te implementeren functionaliteit).

De tests kunnen worden uitgevoerd door dubbel te klikken op CalculatorTest.cs en vervolgens de button  voor het uitvoeren in debug mode en button  voor niet uitvoeren in debug mode.

Schrijf hier de eindtijd op wanneer je klaar bent (unit test slaagt):

E. Opdrachtomschrijving applicatie 2: Hoekberekening

Schrijf hier begintijd hier op wanneer je begint met lezen:

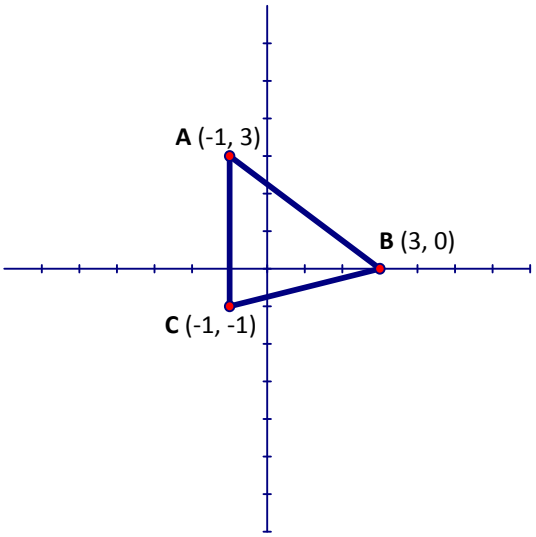
De voorbeeldapplicatie berekent de hoeken van de veelhoeken driehoek en vierhoek. Ten eerste wordt kort uitgelegd hoe de hoeken van een driehoek berekend kunnen worden. Vervolgens wordt uitgelegd hoe de hoeken van een vierhoek berekend kunnen worden, door deze op te splitsen in twee driehoeken.

Domein

Driehoek

Een driehoek bestaat uit de punten A, B en C. In deze voorbeeldapplicatie geldt dat voor elke driehoek het punt A boven punt B ligt en dat punt B boven punt C ligt.

Voorbeeld



Een driehoek heeft de coördinaten (-1,3), (3,0) en (-1,-1). Ten eerste worden de lengte van de zijden berekend en vervolgens worden de hoeken bepaald.

Berekening lengtes van de zijden op basis van coördinaten

De eerste stap is om de lengtes van de zijden te berekenen.

$$\text{afstand A en B} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

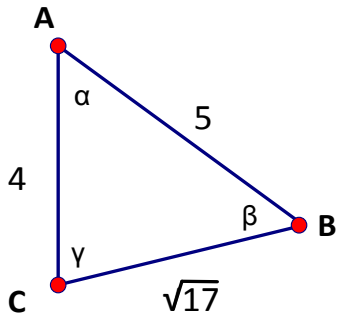
Via deze formule kunnen de volgende lengtes voor de zijden worden gevonden:

$$\text{afstand A en B} = \sqrt{(-1-3)^2 + (3-0)^2} = \sqrt{(-4)^2 + (3)^2} = \sqrt{16+9} = \sqrt{25} = 5$$

$$\text{afstand B en C} = \sqrt{(3--1)^2 + (0--1)^2} = \sqrt{(4)^2 + (1)^2} = \sqrt{16+1} = \sqrt{17} \approx 4{,}12$$

$$\text{afstand A en C} = \sqrt{(-1--1)^2 + (3--1)^2} = \sqrt{(0)^2 + (4)^2} = \sqrt{0+16} = \sqrt{16} = 4$$

Berekening hoeken op basis van de lengtes van de zijden



De hoeken α, β en γ zijn hierboven weergegeven. In de voorbeeldapplicatie geldt voor eenduidigheid dat de bovenste hoek is altijd α, de middelste hoek altijd β en de onderste hoek altijd γ.

Wanneer de lengte van alle zijden bekend zijn, dan kan met behulp van de cosinusregel alle hoeken berekend worden. Voor de cosinusregel krijgen de zijden de namen a, b en c:

- Zijde a is de tegenovergestelde zijde van hoek α. In het voorbeeld geldt dus dat zijde BC als letter a geldt in de cosinusregel.
- Zijde b is de tegenovergestelde zijde van hoek β. In het voorbeeld geldt dus dat zijde AC als letter b geldt in de cosinusregel.
- Zijde c is de tegenovergestelde zijde van hoek γ. In het voorbeeld geldt dus dat zijde AB als letter c geldt in de cosinusregel.

Invullen van de eerder berekende zijden in de cosinusregel levert de gevraagde hoeken op:

$$\text{Hoek A (in graden)} = \cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2 \cdot b \cdot c}\right) = \cos^{-1}\left(\frac{4^2 + 5^2 - \sqrt{17}^2}{2 \cdot 4 \cdot 5}\right) = \cos^{-1}\left(\frac{24}{40}\right) = 53{,}1$$

$$\text{Hoek B (in graden)} = \cos^{-1}\left(\frac{a^2 + c^2 - b^2}{2 \cdot a \cdot c}\right) = \cos^{-1}\left(\frac{\sqrt{17}^2 + 5^2 - 4^2}{2 \cdot \sqrt{17} \cdot 5}\right) = \cos^{-1}\left(\frac{26}{41{,}23}\right) = 50{,}9$$

$$\text{Hoek C (in graden)} = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2 \cdot a \cdot b}\right) = \cos^{-1}\left(\frac{\sqrt{17}^2 + 4^2 - 5^2}{2 \cdot \sqrt{17} \cdot 4}\right) = \cos^{-1}\left(\frac{8}{8 \cdot \sqrt{17}}\right) = 76{,}0$$

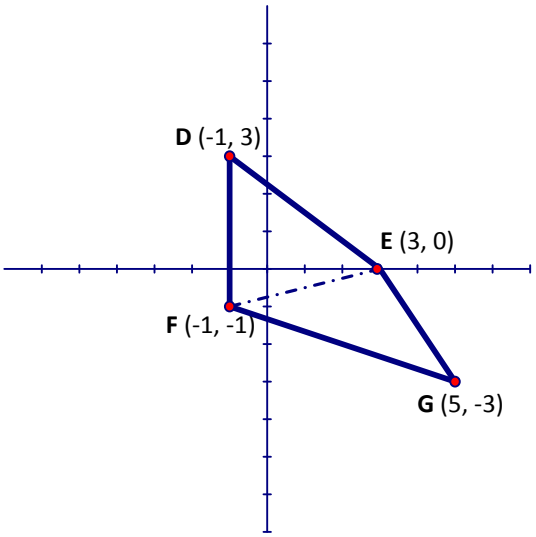
OPMERKING: DE BOVENSTAANDE FORMULES HOEVEN NIET ONTHOUDEN TE WORDEN EN STAAN HIER ALLEEN OM EEN BEELD TE GEVEN HOE DE HOEKEN BEREKEND WORDEN.

Vierhoek

Een driehoek bestaat uit de punten D, E, F en G. In deze voorbeeldapplicatie geldt dat voor elke vierhoek:

- Punt D ligt boven punt E
- Punt E ligt boven punt F
- Punt F ligt boven punt G

Voorbeeld



Een vierhoek heeft de coördinaten (-1,3), (3,0), (-1,-1) en (-5,3).

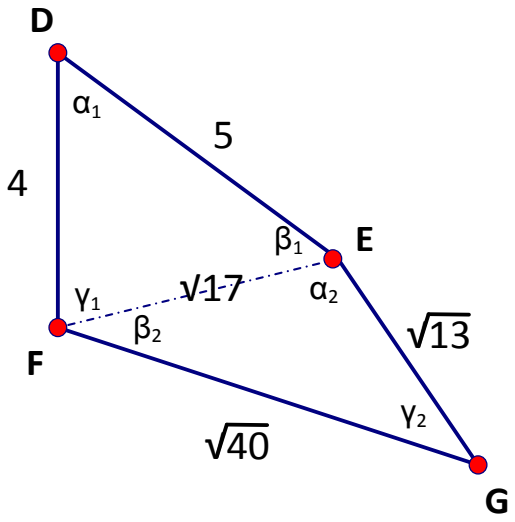
Berekening lengtes van de zijden op basis van coördinaten

Voor het gemak komen dus de punten D, E, F overeen met de punten A, B, C van de driehoek uit het eerdere voorbeeld. De lengtes van de onbekende zijden zijn:

$$\text{afstand E en G} = \sqrt{(3-5)^2 + (0--3)^2} = \sqrt{(-2)^2 + (3)^2} = \sqrt{4+9} = \sqrt{13} \approx 3{,}61$$

$$\text{afstand F en G} = \sqrt{(-1-5)^2 + (-1--3)^2} = \sqrt{(-6)^2 + (2)^2} = \sqrt{36+4} = \sqrt{40} \approx 6{,}32$$

Berekening hoeken op basis van de lengtes van de zijden



De hoeken in de vierhoek kunnen berekend worden door deze op te splitsen in twee driehoeken. De hoeken van de vierhoek zijn dan af te leiden uit de hoeken van de twee driehoeken.

De hoeken van de driehoeken worden berekend via de cosinusregel, zoals in het voorbeeld van de driehoek al is uitgewerkt.

Hoeken van D en G zijn gelijk aan een enkele hoek van een driehoek:

$$\text{Hoek D (in graden)} = \alpha_1 = 53{,}1$$

$$\text{Hoek G (in graden)} = \gamma_2 = 37{,}9$$

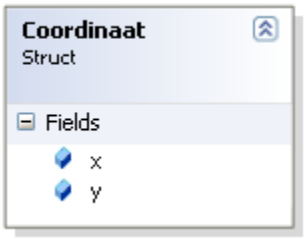
Hoeken van E en F zijn gelijk aan de som van twee hoeken:

$$\text{Hoek E (in graden)} = \beta_1 + \alpha_2 = 50{,}9 + 109{,}7 = 160{,}6$$

$$\text{Hoek F (in graden)} = \gamma_1 + \beta_2 = 76{,}0 + 32{,}5 = 108{,}4$$

Applicatie

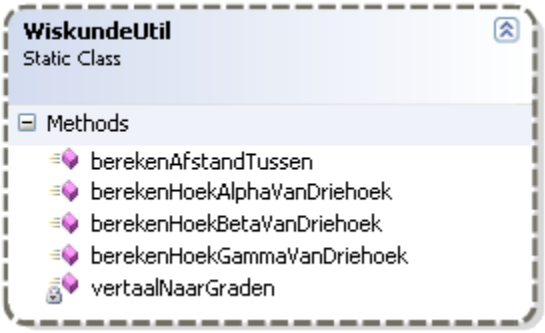
Struct Coördinaat



Figuur 30 De struct Coordinaat houdt simpelweg de x en y coordinaat bij

Een coördinaat is in de applicatie een struct. Het bevat simpelweg de x- en y-coördinaat van een punt in een veelhoek.

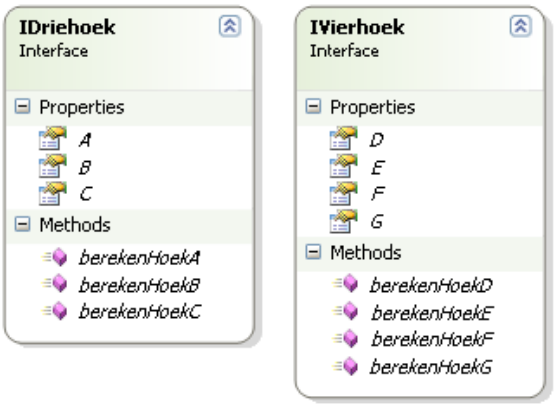
Klasse WiskundeUtil



Figuur 31 De statische WiskundeUtil klasse bevat hulpmethoden

In Figuur 31 staat de statische klasse WiskundeUtil weergegeven. Deze bevat hulpmethoden voor het berekenen van de afstand tussen twee coördinaten en het berekenen van de hoeken α (Alpha), β (Beta) en γ (Gamma) op basis van de cosinusregel. De methode vertaalNaarGraden is private en hoeft verder dus niet gebruikt te worden in de opdracht.

Interfaces IDriehoek en IVierhoek



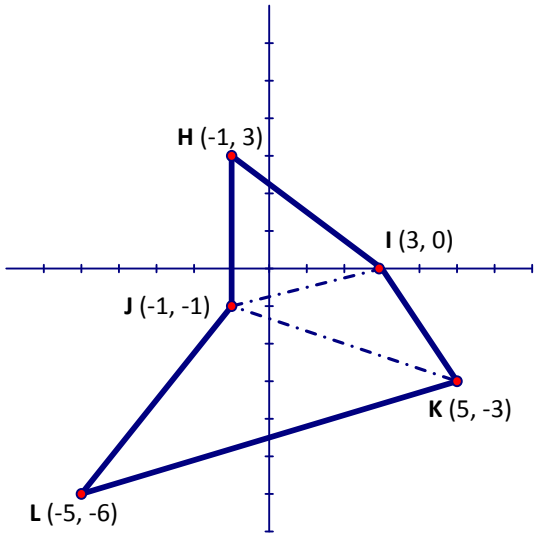
Figuur 32 De interfaces Driehoek en Vierhoek

De interfaces bevatten de properties- en methodendefinities voor het berekenen van een specifieke veelhoek. De punten zijn opgenomen als Properties in de interfaces, welke tijdens het initialiseren gebruikt moeten worden.

Implementatie van de interfaces

De klassen die de interfaces implementeren zijn in het belang van het experiment niet opgenomen.

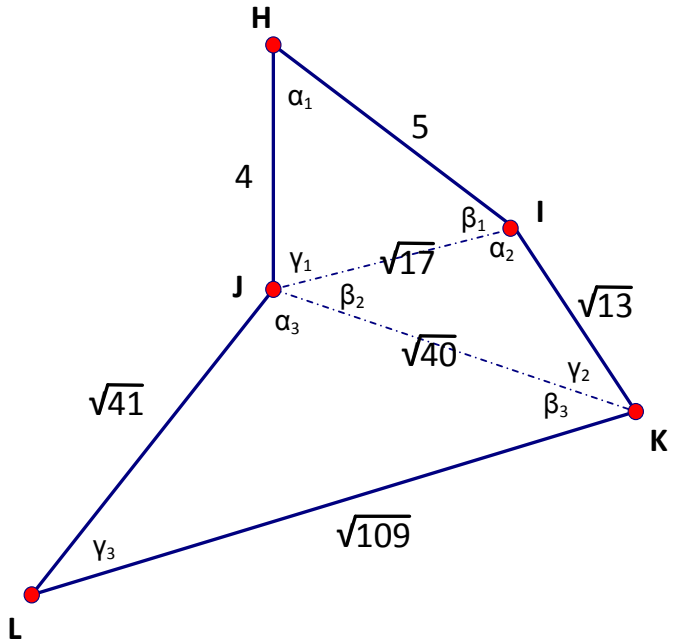
Probleemstelling



Gegeven is nu vijfhoek HIJKL. De punten H, I, J en K komen overeen met respectievelijk punten D, E, F en G uit het vorige voorbeeld. Punt L ligt onder punt K en heeft de coördinaat (-5,-6). De bedoeling is nu om hoek J te berekenen.

1. Bereken de afstand van de lijnen HI, HJ, IJ, JK, JL en KL op basis van de gegeven coördinaten.

In de volgende figuur staan de berekende lengtes weergegeven:



2. Bereken $\boldsymbol{\gamma_1}$ op basis van de lengte van de lijnen HI, HJ, en IJ.

   - HI = 5. Dit is de tegenoverliggende zijde van hoek γ, dus letter c voor de cosinusregel.
   - HJ = 4. Dit is de tegenoverliggende zijde van hoek β, dus letter b voor de cosinusregel.
   - IJ = $\sqrt{17}$. Dit is de tegenoverliggende zijde van hoek α, dus de letter a voor de cosinusregel.

   Invullen van de waarden a, b en c in de cosinusregel levert $\boldsymbol{\gamma_1}\ (in\ graden) = 75{,}96$

3. Bereken $\boldsymbol{\beta_2}$ op basis van de lengte van de lijnen IJ, IK en JK.

   - IJ = $\sqrt{17}$. Dit is de tegenoverliggende zijde van hoek γ, dus letter c voor de cosinusregel.
   - IK = $\sqrt{13}$. Dit is de tegenoverliggende zijde van hoek β, dus letter b voor de cosinusregel.
   - JK =$\sqrt{52}$. Dit is de tegenoverliggende zijde van hoek α, dus de letter a voor de cosinusregel.

   Invullen van de waarden a, b en c in de cosinusregel levert $\boldsymbol{\beta_2}\ (in\ graden) = 32{,}47$

4. Bereken $\boldsymbol{\alpha_3}$ op basis van de lengte van de lijnen JK, JL en KL.

   - JK = $\sqrt{40}$. Dit is de tegenoverliggende zijde van hoek γ, dus letter c voor de cosinusregel.
   - JL = $\sqrt{41}$. Dit is de tegenoverliggende zijde van hoek β, dus letter b voor de cosinusregel.
   - KL =$\sqrt{109}$. Dit is de tegenoverliggende zijde van hoek α, dus de letter a voor de cosinusregel.

   Invullen van de waarden a, b en c in de cosinusregel levert $\boldsymbol{\alpha_3}\ (in\ graden) = 110{,}22$

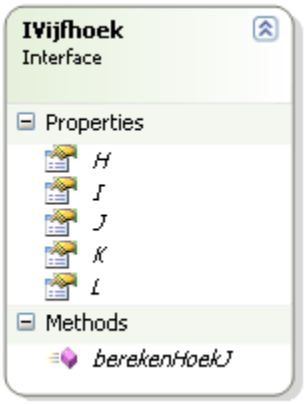
5. Optelling van $\boldsymbol{\gamma_1}$, $\boldsymbol{\beta_2}$ en $\boldsymbol{\alpha_3}$ levert hoek J op.

$$\text{Hoek J (in graden)} = \gamma_1 + \beta_2 + \alpha_3 = 79{,}52 + 32{,}47 + 110{,}22 = 218{,}7$$

Opdrachtformulering

Open de solution “Hoekberekening”.

Merk hierbij op dat de volgende interface IVijfhoek alvast is opgenomen:



Implementeer de methoden uit de interface IVijfhoek in een klasse, zodat hoek J berekend wordt aan de hand van de vijf coördinaten H, I, J, K en L.

Testen

Om te controleren of de aanpassing correct is uitgevoerd, moeten alle unit tests slagen. Deze zijn al gedefinieerd in een apart C# Test Project “HoekberekeningTest”. De testgevallen in de unit tests komen precies overeen met de reeds behandelde voorbeelden.

De test voor hoek J faalt in eerste instantie. Dit komt vanwege de volgende regel in berekenHoekJTest():

```
Assert.Fail("Verwijder deze regel, uncomment deze methode en vul bovenaan
jouw klassenaam in die interface IVijfhoek implementeert");
```

Volg de actie die in de regel beschreven staat op. De unit tests kunnen vervolgens worden uitgevoerd door dubbel te klikken op HoekTest.cs en vervolgens de button  voor het uitvoeren in debug mode en button  voor niet uitvoeren in debug mode.

Schrijf hier de eindtijd op wanneer je klaar bent (unit test slaagt):

F. Opdrachtomschrijving applicatie 3 : Belasting

Schrijf hier begintijd hier op wanneer je begint met lezen:

Domein

Motorrijtuigenbelasting

De motorrijtuigenbelasting of houderschapsbelasting (populair benaming "wegenbelasting") is een belasting die onder meer in Nederland wordt geheven op het bezit van een motorvoertuig. Naast het motorvoertuig (=”trekker”) wordt ook belasting geheven op het bezitten van een oplegger.

Berekening van belasting

De belasting wordt per voertuig (trekker of oplegger) berekend. Hiervoor zijn een belastingfactor en gewichtfactor van toepassing, welke zijn weergegeven in Tabel 53 en Tabel 54.

Voertuig	Belastingfactor
Trekker	75
Oplegger	85

Tabel 53 Belastingfactor per voertuig

Gewicht	Gewichtfactor
Minder of gelijk aan 1500 kg	1.1
Van 1500 tot en met 2250 kg	1.2
Meer dan 2250 kg	1.3

Tabel 54 Gewichtfactor per voertuig

Voor het berekenen van de belasting zijn de volgende gegevens benodigd:

$$Waardevermindering = \frac{kostprijs \cdot (huidig\ jaar - bouwjaar)}{10}$$

$$Huidige\ waarde\ voertuig = kostprijs - waardevermindering$$

De belasting van een trekker of oplegger kan op de volgende manier berekend worden:

$$Belasting = \frac{Huidige\ waarde\ voertuig}{Belastingfactor} \cdot Gewichtfactor$$

De totale belasting is simpelweg de som van de voertuigbelasting van de trekker en de oplegger:

$$Totale\ belasting = Belasting\ trekker + Belasting\ oplegger$$

Applicatie

In het belang van het experiment is er geen beschrijving van de bestaande typen in de bestaande applicatie opgenomen.

Probleemstelling

Er zijn twee veranderingen gekomen in de berekening van de belastingen.

Verandering 1: belasting voor een oplegger

Er is een nieuwe berekening om de belasting te berekenen van de belasting voor een oplegger.

$$Nieuwe\ belasting\ oplegger\ = belasting\ oplegger + toegevoegde\ belasting$$

De toegevoegde belasting staat weergegeven in volgende tabel:

Aantal wielen van oplegger	Toegevoegde belasting
Minder of gelijk aan 4	Geen toegevoegde belasting
Van 5 tot en met 8	5% van vorige belasting
Meer dan 8	10% van vorige belasting

Tabel 55 Toegevoegde belasting voor een oplegger

Verandering 2: totale belasting

De totale voertuigbelasting is niet simpelweg meer een som, maar het bevat nu ook een bijbetaling aan de hand van de zone waar de combinatie van een trekker-oplegger is geregistreerd.

Zone	Toeslagfactor
1	1,05
2	1,08
3	1,10
Elke andere zone	1,12

Tabel 56 Toegevoegde belasting voor een oplegger

De nieuwe formule luidt als volgt:

$$Totale\ belasting\ = (belasting\ trekker + \ nieuwe\ belasting\ oplegger) * toeslagfactor$$

In de formule valt te zien dat de toeslagfactor dus over de nieuwe belasting van de oplegger berekend wordt, zoals in verandering 1 is beschreven.

Opdrachtformulering

Open de solution “Belasting”.

Implementeer de beschreven verandering 1 en verandering 2 in de bestaande applicatie.

In deze opdracht is benodigd om nieuwe velden toe te voegen voor het aantal wielen van de oplegger (verandering 1) en de zone van de combinatie van een trekker-oplegger (verandering 2).

Om te controleren of de functionaliteit correct geïmplementeerd is, is er een Test Project “BelastingTest” beschikbaar. In deze test moeten eerst de nieuwe toegevoegde velden worden geïnitialiseerd (zie de opgenomen TODO’s).

Schrijf hier de eindtijd op wanneer je klaar bent (unit test slaagt):

G. Broncode applicatie 1 : Levensverzekering

Identiek voor versies A en B

Constanten.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Verzekeringen
{
    public class Constanten
    {
        public const int MAANDEN_IN_JAAR = 12;

        public const double WINSTFACTOR = 2.0;
        public const double KORTINGSFACTOR = 0.96;
        public const double ROKERSFACTOR = 1.10;

        public const int GEMIDDELDE_LOOPTIJD_KORT = 8;
        public const int GEMIDDELDE_LOOPTIJD_GEMIDDELD = 23;
        public const int GEMIDDELDE_LOOPTIJD_LANG = 38;

        public const double GEMIDDELD_PERCENTAGE_VOLBRACHTE_LOOPTIJD_BIJ_OVERLIJDEN = 0.50;
    }
}
```

Levensverzekering.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Verzekeringen
{
    public enum LooptijdKlasse
    {
        Begin = 0,
        Kort = 0, // looptijd tussen 0 en 15 jaar
        Gemiddeld = 1, // looptijd tussen 15 en 30 jaar
        Lang = 2, // looptijd tussen 30 en 45 jaar
        Einde = 2
    }

    public class Levensverzekering
    {
        private LooptijdKlasse _looptijd;
        private double _verzekerdeBedrag;

        public LooptijdKlasse Looptijd
        {
            get { return _looptijd; }
            set { _looptijd = value; }
        }

        public double VerzekerdeBedrag
        {
            get { return _verzekerdeBedrag; }
            set { _verzekerdeBedrag = value; }
        }

    }
}
```

Persoon.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Verzekeringen
{
    public enum LeeftijdKlasse
    {
        Begin = 0,
        Junior = 0, // leeftijd tussen 20 en 35
        Medior = 1, // leeftijd tussen 35 en 50
        Senior = 2, // leeftijd tussen 50 en 65
        Einde = 2,
    }

    public class Persoon
    {
        private LeeftijdKlasse _leeftijd;
        private bool _isRoker;
        private bool _isBestaandeKlant;

        public LeeftijdKlasse Leeftijd
        {
            get { return _leeftijd; }
            set { _leeftijd = value; }
        }

        public bool IsRoker
        {
            get { return _isRoker; }
            set { _isRoker = value; }
        }

        public bool IsBestaandeKlant
        {
            get { return _isBestaandeKlant; }
            set { _isBestaandeKlant = value; }
        }
    }
}
```

ICalculator.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Verzekeringen
{
    public interface ICalculator
    {
         double berekenPremiePerMaand(Persoon persoon, Levensverzekering verzekering);
    }
}
```

Versie A

Calculator.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Verzekeringen
{
    public class Calculator : ICalculator
    {
        private double[,] sterfteKansTabel;
```

```
        public Calculator()
        {
            int aantalLeeftijdsKlassen = (int)LeeftijdKlasse.Einde;
            int aantalLooptijdDuren = (int)LooptijdKlasse.Einde;

            sterfteKansTabel = new double[aantalLeeftijdsKlassen + 1, aantalLooptijdDuren + 1];

            initialiseerSterfteKansTabel();
        }

        public void initialiseerSterfteKansTabel()
        {
            int junior = (int)LeeftijdKlasse.Junior;
            int medior = (int)LeeftijdKlasse.Medior;
            int senior = (int)LeeftijdKlasse.Senior;

            int kort = (int)LooptijdKlasse.Kort;
            int gemiddeld = (int)LooptijdKlasse.Gemiddeld;
            int lang = (int)LooptijdKlasse.Lang;

            sterfteKansTabel[junior, kort] = 0.05;
            sterfteKansTabel[junior, gemiddeld] = 0.30;
            sterfteKansTabel[junior, lang] = 0.55;
            sterfteKansTabel[medior, kort] = 0.23;
            sterfteKansTabel[medior, gemiddeld] = 0.47;
            sterfteKansTabel[medior, lang] = 0.71;
            sterfteKansTabel[senior, kort] = 0.40;
            sterfteKansTabel[senior, gemiddeld] = 0.81;
            sterfteKansTabel[senior, lang] = 0.98;
        }

        public double berekenPremiePerMaand(Persoon persoon, Levensverzekering verzekering)
        {
            int leeftijdsKlasse = (int)persoon.Leeftijd;
            int looptijdKlasse = (int)verzekering.Looptijd;

            double sterftekans = sterfteKansTabel[leeftijdsKlasse, looptijdKlasse];

            double betaalBedragZonderWinst = berekenBetaalBedragZonderWinst(sterftekans,
verzekering.VerzekerdeBedrag);

            double betaalBedragMetWinst = betaalBedragZonderWinst * Constanten.WINSTFACTOR;

            double premiePerMaand = berekenBedragPerMaand(betaalBedragMetWinst,
verzekering.Looptijd);

            if (persoon.IsBestaandeKlant)
            {
                premiePerMaand = premiePerMaand * Constanten.KORTINGSFACTOR;
            }

            return premiePerMaand;
        }

        private double berekenBedragPerMaand(double bedrag, LooptijdKlasse looptijd)
        {
            int gemiddeldeLooptijdDuur = 0;
            switch (looptijd)
            {
                case LooptijdKlasse.Kort:
                    gemiddeldeLooptijdDuur = Constanten.GEMIDDELDE_LOOPTIJD_KORT;
                    break;
                case LooptijdKlasse.Gemiddeld:
                    gemiddeldeLooptijdDuur = Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD;
                    break;
                case LooptijdKlasse.Lang:
                    gemiddeldeLooptijdDuur = Constanten.GEMIDDELDE_LOOPTIJD_LANG;
                    break;
            }
```

```
            int aantalMaanden = gemiddeldeLooptijdDuur * Constanten.MAANDEN_IN_JAAR;
            double bedragPerMaand = bedrag / aantalMaanden;

            return bedragPerMaand;
        }

        private double berekenBetaalBedragZonderWinst(double sterftekans, double verzekerdBedrag)
        {
            return (sterftekans * verzekerdBedrag) /
            (1 - (1 - Constanten.GEMIDDELD_PERCENTAGE_VOLBRACHTE_LOOPTIJD_BIJ_OVERLIJDEN) *
sterftekans);
        }
    }
}
```

Versie B

Calculator.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Verzekeringen
{
    public class Calculator : ICalculator
    {
        public double berekenPremiePerMaand(Persoon persoon, Levensverzekering verzekering)
        {
            double sterftekans = 0.0;

            if (persoon.Leeftijd == LeeftijdKlasse.Junior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Kort)
                {
                    sterftekans = 0.05;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Junior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Gemiddeld)
                {
                    sterftekans = 0.30;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Junior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Lang)
                {
                    sterftekans = 0.55;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Medior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Kort)
                {
                    sterftekans = 0.23;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Medior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Gemiddeld)
                {
                    sterftekans = 0.47;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Medior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Lang)
```

```
                {
                    sterftekans = 0.71;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Senior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Kort)
                {
                    sterftekans = 0.40;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Senior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Gemiddeld)
                {
                    sterftekans = 0.81;
                }
            }
            if (persoon.Leeftijd == LeeftijdKlasse.Senior)
            {
                if (verzekering.Looptijd == LooptijdKlasse.Lang)
                {
                    sterftekans = 0.98;
                }
            }

            double betaalBedragZonderWinst = berekenBetaalBedragZonderWinst(sterftekans,
verzekering.VerzekerdeBedrag);

            double betaalBedragMetWinst = betaalBedragZonderWinst * Constanten.WINSTFACTOR;

            int gemiddeldeLooptijdDuur = 0;
            switch(verzekering.Looptijd)
            {
                case LooptijdKlasse.Kort :
                    gemiddeldeLooptijdDuur = Constanten.GEMIDDELDE_LOOPTIJD_KORT;
                    break;
                case LooptijdKlasse.Gemiddeld:
                    gemiddeldeLooptijdDuur = Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD;
                    break;
                case LooptijdKlasse.Lang:
                    gemiddeldeLooptijdDuur = Constanten.GEMIDDELDE_LOOPTIJD_LANG;
                    break;
            }

            int aantalMaanden = gemiddeldeLooptijdDuur * Constanten.MAANDEN_IN_JAAR;
            double premiePerMaand = betaalBedragMetWinst / aantalMaanden;

            if (persoon.IsBestaandeKlant)
            {
                premiePerMaand = premiePerMaand * Constanten.KORTINGSFACTOR;
            }

            return premiePerMaand;
        }

        private double berekenBetaalBedragZonderWinst(double sterftekans, double verzekerdBedrag)
        {
            return (sterftekans * verzekerdBedrag) /
            (1 - (1 - Constanten.GEMIDDELD_PERCENTAGE_VOLBRACHTE_LOOPTIJD_BIJ_OVERLIJDEN) *
sterftekans);
        }
    }
}
```

Automatische test (identiek voor versie A en B)

CalculatorTest.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Text;
using System.Collections.Generic;
using InfoSupport.Experiment.Verzekeringen;

namespace InfoSupport.Experiment.Verzekeringen
{
    [TestClass()]
    public class CalculatorTest
    {
        private TestContext testContextInstance;

        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }

        [TestMethod()]
        public void berekenPremiePerMaandHuidigeFunctionaliteitTest()
        {
            ICalculator target = new Calculator();

            Persoon persoon = new Persoon();
            Levensverzekering verzekering = new Levensverzekering();
            double premiePerMaandVerwacht;
            double premiePerMaand;

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Junior;
            persoon.IsBestaandeKlant = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 154678.0;
            verzekering.Looptijd = LooptijdKlasse.Kort;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.05,
verzekering.VerzekerdeBedrag) *
                                     Constanten.WINSTFACTOR /
                                      (
                                          Constanten.GEMIDDELDE_LOOPTIJD_KORT *
                                          Constanten.MAANDEN_IN_JAAR
                                      ) *
                                      Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Junior;
            persoon.IsBestaandeKlant = false;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 999999.0;
            verzekering.Looptijd = LooptijdKlasse.Gemiddeld;
```

```
            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.30,
verzekering.VerzekerdeBedrag) /
                                        (
                                            Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD *
                                            Constanten.MAANDEN_IN_JAAR
                                        ) *
                                        Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Junior;
            persoon.IsBestaandeKlant = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 50050.0;
            verzekering.Looptijd = LooptijdKlasse.Lang;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.55,
verzekering.VerzekerdeBedrag) /
                                        (
                                            Constanten.GEMIDDELDE_LOOPTIJD_LANG *
                                            Constanten.MAANDEN_IN_JAAR
                                        ) *
                                        Constanten.WINSTFACTOR *
                                        Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Medior;
            persoon.IsBestaandeKlant = false;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 100000000.0;
            verzekering.Looptijd = LooptijdKlasse.Kort;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.23,
verzekering.VerzekerdeBedrag) /
                                        (
                                            Constanten.GEMIDDELDE_LOOPTIJD_KORT *
                                            Constanten.MAANDEN_IN_JAAR
                                        ) *
                                        Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Medior;
            persoon.IsBestaandeKlant = true;
```

```
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 60000.0;
            verzekering.Looptijd = LooptijdKlasse.Gemiddeld;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.47,
verzekering.VerzekerdeBedrag) /
                                     (
                                         Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD *
                                         Constanten.MAANDEN_IN_JAAR
                                     ) *
                                     Constanten.WINSTFACTOR *
                                     Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Medior;
            persoon.IsBestaandeKlant = false;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 120000.0;
            verzekering.Looptijd = LooptijdKlasse.Lang;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.71,
verzekering.VerzekerdeBedrag) /
                                     (
                                         Constanten.GEMIDDELDE_LOOPTIJD_LANG *
                                         Constanten.MAANDEN_IN_JAAR
                                     ) *
                                     Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Senior;
            persoon.IsBestaandeKlant = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 30000.0;
            verzekering.Looptijd = LooptijdKlasse.Kort;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.40,
verzekering.VerzekerdeBedrag) /
                                     (
                                         Constanten.GEMIDDELDE_LOOPTIJD_KORT *
                                         Constanten.MAANDEN_IN_JAAR
                                     ) *
                                     Constanten.WINSTFACTOR *
                                     Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );
```

```
            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Senior;
            persoon.IsBestaandeKlant = false;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 50000.0;
            verzekering.Looptijd = LooptijdKlasse.Gemiddeld;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.81,
verzekering.VerzekerdeBedrag) /
                                        (
                                            Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD *
                                            Constanten.MAANDEN_IN_JAAR
                                        ) *
                                        Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Senior;
            persoon.IsBestaandeKlant = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 50000.0;
            verzekering.Looptijd = LooptijdKlasse.Lang;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.98,
verzekering.VerzekerdeBedrag) /
                                        (
                                            Constanten.GEMIDDELDE_LOOPTIJD_LANG *
                                            Constanten.MAANDEN_IN_JAAR
                                        ) *
                                        Constanten.WINSTFACTOR *
                                        Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );
        }

        [TestMethod()]
        public void berekenPremiePerMaandNieuweFunctionaliteitTest()
        {
            ICalculator target = new Calculator();

            Persoon persoon = new Persoon();
            Levensverzekering verzekering = new Levensverzekering();
            double premiePerMaandVerwacht;
            double premiePerMaand;

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Junior;
            persoon.IsBestaandeKlant = true;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 154678.0;
            verzekering.Looptijd = LooptijdKlasse.Kort;
```

```
            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.07,
verzekering.VerzekerdeBedrag) *
                                     Constanten.ROKERSFACTOR *
                                     Constanten.WINSTFACTOR /
                                      (
                                          Constanten.GEMIDDELDE_LOOPTIJD_KORT *
                                          Constanten.MAANDEN_IN_JAAR
                                      ) *
                                      Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Junior;
            persoon.IsBestaandeKlant = false;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 999999.0;
            verzekering.Looptijd = LooptijdKlasse.Gemiddeld;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.42,
verzekering.VerzekerdeBedrag) *
                                     Constanten.ROKERSFACTOR /
                                     (
                                         Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD *
                                         Constanten.MAANDEN_IN_JAAR
                                     ) *
                                     Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Junior;
            persoon.IsBestaandeKlant = true;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 50050.0;
            verzekering.Looptijd = LooptijdKlasse.Lang;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.77,
verzekering.VerzekerdeBedrag)*
                                     Constanten.ROKERSFACTOR /
                                     (
                                         Constanten.GEMIDDELDE_LOOPTIJD_LANG *
                                         Constanten.MAANDEN_IN_JAAR
                                     ) *
                                     Constanten.WINSTFACTOR *
                                     Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
```

```
                        );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Medior;
            persoon.IsBestaandeKlant = false;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 100000000.0;
            verzekering.Looptijd = LooptijdKlasse.Kort;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.29,
verzekering.VerzekerdeBedrag)*
                                    Constanten.ROKERSFACTOR /
                                    (
                                        Constanten.GEMIDDELDE_LOOPTIJD_KORT *
                                        Constanten.MAANDEN_IN_JAAR
                                    ) *
                                    Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Medior;
            persoon.IsBestaandeKlant = true;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 60000.0;
            verzekering.Looptijd = LooptijdKlasse.Gemiddeld;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.59,
verzekering.VerzekerdeBedrag) *
                                    Constanten.ROKERSFACTOR /
                                    (
                                        Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD *
                                        Constanten.MAANDEN_IN_JAAR
                                    ) *
                                    Constanten.WINSTFACTOR *
                                    Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Medior;
            persoon.IsBestaandeKlant = false;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 120000.0;
            verzekering.Looptijd = LooptijdKlasse.Lang;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.89,
verzekering.VerzekerdeBedrag) *
                                    Constanten.ROKERSFACTOR /
                                    (
                                        Constanten.GEMIDDELDE_LOOPTIJD_LANG *
                                        Constanten.MAANDEN_IN_JAAR
                                    ) *
```

```
                                    Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Senior;
            persoon.IsBestaandeKlant = true;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 30000.0;
            verzekering.Looptijd = LooptijdKlasse.Kort;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.44,
verzekering.VerzekerdeBedrag) *
                                    Constanten.ROKERSFACTOR /
                                    (
                                        Constanten.GEMIDDELDE_LOOPTIJD_KORT *
                                        Constanten.MAANDEN_IN_JAAR
                                    ) *
                                    Constanten.WINSTFACTOR *
                                    Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Senior;
            persoon.IsBestaandeKlant = false;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 50000.0;
            verzekering.Looptijd = LooptijdKlasse.Gemiddeld;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.89,
verzekering.VerzekerdeBedrag)*
                                    Constanten.ROKERSFACTOR /
                                    (
                                        Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD *
                                        Constanten.MAANDEN_IN_JAAR
                                    ) *
                                    Constanten.WINSTFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );

            persoon = new Persoon();
            persoon.Leeftijd = LeeftijdKlasse.Senior;
            persoon.IsBestaandeKlant = true;
            persoon.IsRoker = true;
            verzekering = new Levensverzekering();
            verzekering.VerzekerdeBedrag = 50000.0;
```

```
            verzekering.Looptijd = LooptijdKlasse.Lang;

            premiePerMaandVerwacht = berekenBetaalBedragZonderWinst(0.99,
verzekering.VerzekerdeBedrag)*
                                        Constanten.ROKERSFACTOR /
                                        (
                                            Constanten.GEMIDDELDE_LOOPTIJD_LANG *
                                            Constanten.MAANDEN_IN_JAAR
                                        ) *
                                        Constanten.WINSTFACTOR *
                                        Constanten.KORTINGSFACTOR;

            premiePerMaand = target.berekenPremiePerMaand(persoon, verzekering);

            Assert.AreEqual(
                                premiePerMaandVerwacht,
                                premiePerMaand,
                                "Methode berekenPremiePerMaand geeft " + premiePerMaand +
                                ". Verwacht: " + premiePerMaandVerwacht + "."
                            );
        }

        private double berekenBetaalBedragZonderWinst(double sterftekans, double verzekerdBedrag)
        {
            return (sterftekans * verzekerdBedrag) /
            (1 - (1 - Constanten.GEMIDDELD_PERCENTAGE_VOLBRACHTE_LOOPTIJD_BIJ_OVERLIJDEN) *
sterftekans);
        }
    }
}
```

H. Broncode applicatie 2 : Hoekberekening

Identieke code voor versies A en B

Coordinaat.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Hoekberekening
{
    public struct Coordinaat
    {
        public int x;
        public int y;

        public Coordinaat(int x, int y)
        {
            this.x = x;
            this.y = y;
        }
    }
}
```

WiskundeUtil.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Hoekberekening
{
    public static class WiskundeUtil
    {
        public static double berekenHoekAlphaVanDriehoek(double a, double b, double c)
        {
            return vertaalNaarGraden(Math.Acos((Math.Pow(b, 2) + Math.Pow(c, 2) - Math.Pow(a, 2))
/ (2 * b * c)));
        }

        public static double berekenHoekBetaVanDriehoek(double a, double b, double c)
        {
            return vertaalNaarGraden(Math.Acos((Math.Pow(a, 2) + Math.Pow(c, 2) - Math.Pow(b, 2))
/ (2 * a * c)));
        }

        public static double berekenHoekGammaVanDriehoek(double a, double b, double c)
        {
            return vertaalNaarGraden(Math.Acos((Math.Pow(a, 2) + Math.Pow(b, 2) - Math.Pow(c, 2))
/ (2 * a * b)));
        }

        public static double berekenAfstandTussen(Coordinaat c1, Coordinaat c2)
        {
            return Math.Sqrt( Math.Pow(c1.x - c2.x, 2) +  Math.Pow(c1.y - c2.y, 2));
        }

        private static double vertaalNaarGraden(double radialen)
        {
            return radialen * 180 / Math.PI;
        }
    }
}
```

IDriehoek.cs

```
using System;
using System.Collections.Generic;
using System.Text;
namespace InfoSupport.Experiment.Hoekberekening
{
    public interface IDriehoek
    {
        Coordinaat A
        {
            set;
        }

        Coordinaat B
        {
            set;
        }

        Coordinaat C
        {
            set;
        }

        double berekenHoekA();

        double berekenHoekB();

        double berekenHoekC();
    }
}
```

IVierhoek.cs

```
using System;
using System.Collections.Generic;
using System.Text;
namespace InfoSupport.Experiment.Hoekberekening
{
    public interface IVierhoek
    {
        Coordinaat D
        {
            set;
        }

        Coordinaat E
        {
            set;
        }

        Coordinaat F
        {
            set;
        }

        Coordinaat G
        {
            set;
        }

        double berekenHoekD();

        double berekenHoekE();

        double berekenHoekF();

        double berekenHoekG();
    }
}
```

IVijfhoek.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Hoekberekening
{
    public interface IVijfhoek
    {
        Coordinaat H
        {
            set;
        }

        Coordinaat I
        {
            set;
        }

        Coordinaat J
        {
            set;
        }

        Coordinaat K
        {
            set;
        }

        Coordinaat L
        {
            set;
        }

        double berekenHoekJ();
    }
}
```

Versie A

Veelhoek.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Hoekberekening
{
    public class Veelhoek : IDriehoek, IVierhoek
    {
        private Coordinaat _a;
        private Coordinaat _b;
        private Coordinaat _c;

        private Coordinaat _d;
        private Coordinaat _e;
        private Coordinaat _f;
        private Coordinaat _g;

        public Coordinaat A
        {
            set { _a = value; }
        }

        public Coordinaat B
        {
            set { _b = value; }
        }
```

```
public Coordinaat C
{
    set { _c = value; }
}

public Coordinaat D
{
    set { _d = value; }
}

public Coordinaat E
{
    set { _e = value; }
}

public Coordinaat F
{
    set { _f = value; }
}

public Coordinaat G
{
    set { _g = value; }
}

public double berekenHoekA()
{
    //Berekening lengtes van de zijden op basis van coördinaten
    double ab = WiskundeUtil.berekenAfstandTussen(_a, _b);
    double bc = WiskundeUtil.berekenAfstandTussen(_b, _c);
    double ac = WiskundeUtil.berekenAfstandTussen(_a, _c);

    //Koppeling zijden aan de letters a, b, c voor cosinusregel
    double a = bc;
    double b = ac;
    double c = ab;

    //Berekening hoek op basis van lengtes van de zijden
    double hoekA = WiskundeUtil.berekenHoekAlphaVanDriehoek(a, b, c);

    return hoekA;
}

public double berekenHoekB()
{
    //Berekening lengtes van de zijden op basis van coördinaten
    double ab = WiskundeUtil.berekenAfstandTussen(_a, _b);
    double bc = WiskundeUtil.berekenAfstandTussen(_b, _c);
    double ac = WiskundeUtil.berekenAfstandTussen(_a, _c);

    //Koppeling zijden aan de letters a, b, c voor cosinusregel
    double a = bc;
    double b = ac;
    double c = ab;

    double hoekB = WiskundeUtil.berekenHoekBetaVanDriehoek(a, b, c);

    return hoekB;
}

public double berekenHoekC()
{
    //Berekening lengtes van de zijden op basis van coördinaten
    double ab = WiskundeUtil.berekenAfstandTussen(_a, _b);
    double bc = WiskundeUtil.berekenAfstandTussen(_b, _c);
    double ac = WiskundeUtil.berekenAfstandTussen(_a, _c);

    //Koppeling zijden aan de letters a, b, c voor cosinusregel
    double a = bc;
    double b = ac;
```

```
        double c = ab;

        double hoekC = WiskundeUtil.berekenHoekGammaVanDriehoek(a, b, c);

        return hoekC;
    }

    public double berekenHoekD()
    {
        //Berekening lengtes van de zijden op basis van coördinaten
        double de = WiskundeUtil.berekenAfstandTussen(_d, _e);
        double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
        double df = WiskundeUtil.berekenAfstandTussen(_d, _f);

        //Zijden van driehoek 2 zijn n.v.t. voor het berekenen van hoek D

        //Koppeling zijden aan de letters a, b, c voor cosinusregel
        double a = ef;
        double b = df;
        double c = de;

        double alpha1 =  WiskundeUtil.berekenHoekAlphaVanDriehoek(a, b, c);

        double hoekD = alpha1;

        return hoekD;
    }

    public double berekenHoekE()
    {
        //Berekening lengtes van de zijden van driehoek 1 op basis van coördinaten
        double de = WiskundeUtil.berekenAfstandTussen(_d, _e);
        double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
        double df = WiskundeUtil.berekenAfstandTussen(_d, _f);

        //Koppeling zijden aan de letters a, b, c voor cosinusregel
        double a = ef;
        double b = df;
        double c = de;

        double beta1 = WiskundeUtil.berekenHoekBetaVanDriehoek(a, b, c);

        //Berekening lengtes van de zijden van driehoek 2 op basis van coördinaten
        //ef is al berekend
        double fg = WiskundeUtil.berekenAfstandTussen(_f, _g);
        double eg = WiskundeUtil.berekenAfstandTussen(_e, _g);

        //Koppeling zijden aan de letters a, b, c voor cosinusregel
        a = fg;
        b = eg;
        c = ef;

        double alpha2 = WiskundeUtil.berekenHoekAlphaVanDriehoek(a, b, c);

        double hoekE = beta1 + alpha2;

        return hoekE;
    }

    public double berekenHoekF()
    {
        //Berekening lengtes van de zijden van driehoek 1 op basis van coördinaten
        double de = WiskundeUtil.berekenAfstandTussen(_d, _e);
        double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
        double df = WiskundeUtil.berekenAfstandTussen(_d, _f);

        //Koppeling zijden aan de letters a, b, c voor cosinusregel
        double a = ef;
        double b = df;
        double c = de;
```

```
            double gamma1 = WiskundeUtil.berekenHoekGammaVanDriehoek(a, b, c);

            //Berekening lengtes van de zijden van driehoek 2 op basis van coördinaten
            //ef is al berekend
            double fg = WiskundeUtil.berekenAfstandTussen(_f, _g);
            double eg = WiskundeUtil.berekenAfstandTussen(_e, _g);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            a = fg;
            b = eg;
            c = ef;

            double beta2 = WiskundeUtil.berekenHoekBetaVanDriehoek(a, b, c);

            double hoekF = gamma1 + beta2;

            return hoekF;
        }

        public double berekenHoekG()
        {
            //Zijden van driehoek 1 zijn n.v.t. voor het berekenen van hoek G

            //Berekening lengtes van de zijden van driehoek 2 op basis van coördinaten
            double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
            double fg = WiskundeUtil.berekenAfstandTussen(_f, _g);
            double eg = WiskundeUtil.berekenAfstandTussen(_e, _g);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            double a = fg;
            double b = eg;
            double c = ef;

            double gamma2 = WiskundeUtil.berekenHoekGammaVanDriehoek(a, b, c);

            double hoekG = gamma2;

            return hoekG;
        }
    }
}
```

Versie B

Driehoek.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Hoekberekening
{
    public class Driehoek : IDriehoek
    {
        private Coordinaat _a;
        private Coordinaat _b;
        private Coordinaat _c;

        public Coordinaat A
        {
            set { _a = value; }
        }

        public Coordinaat B
        {
            set { _b = value; }
        }

        public Coordinaat C
```

```
        {
            set { _c = value; }
        }

        public double berekenHoekA()
        {
            //Berekening lengtes van de zijden op basis van coördinaten
            double ab = WiskundeUtil.berekenAfstandTussen(_a, _b);
            double bc = WiskundeUtil.berekenAfstandTussen(_b, _c);
            double ac = WiskundeUtil.berekenAfstandTussen(_a, _c);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            double a = bc;
            double b = ac;
            double c = ab;

            //Berekening hoek op basis van lengtes van de zijden
            double hoekA = WiskundeUtil.berekenHoekAlphaVanDriehoek(a, b, c);

            return hoekA;
        }

        public double berekenHoekB()
        {
            //Berekening lengtes van de zijden op basis van coördinaten
            double ab = WiskundeUtil.berekenAfstandTussen(_a, _b);
            double bc = WiskundeUtil.berekenAfstandTussen(_b, _c);
            double ac = WiskundeUtil.berekenAfstandTussen(_a, _c);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            double a = bc;
            double b = ac;
            double c = ab;

            double hoekB = WiskundeUtil.berekenHoekBetaVanDriehoek(a, b, c);

            return hoekB;
        }

        public double berekenHoekC()
        {
            //Berekening lengtes van de zijden op basis van coördinaten
            double ab = WiskundeUtil.berekenAfstandTussen(_a, _b);
            double bc = WiskundeUtil.berekenAfstandTussen(_b, _c);
            double ac = WiskundeUtil.berekenAfstandTussen(_a, _c);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            double a = bc;
            double b = ac;
            double c = ab;

            double hoekC = WiskundeUtil.berekenHoekGammaVanDriehoek(a, b, c);

            return hoekC;
        }
    }
}
```

Vierhoek.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Hoekberekening
{
    public class Vierhoek : IVierhoek
    {
        private Coordinaat _d;
        private Coordinaat _e;
        private Coordinaat _f;
```

```
private Coordinaat _g;

public Coordinaat D
{
    set { _d = value; }
}

public Coordinaat E
{
    set { _e = value; }
}

public Coordinaat F
{
    set { _f = value; }
}

public Coordinaat G
{
    set { _g = value; }
}

public double berekenHoekD()
{
    //Berekening lengtes van de zijden op basis van coördinaten
    double de = WiskundeUtil.berekenAfstandTussen(_d, _e);
    double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
    double df = WiskundeUtil.berekenAfstandTussen(_d, _f);

    //Zijden van driehoek 2 zijn n.v.t. voor het berekenen van hoek D

    //Koppeling zijden aan de letters a, b, c voor cosinusregel
    double a = ef;
    double b = df;
    double c = de;

    double alpha1 = WiskundeUtil.berekenHoekAlphaVanDriehoek(a, b, c);

    double hoekD = alpha1;

    return hoekD;
}

public double berekenHoekE()
{
    //Berekening lengtes van de zijden van driehoek 1 op basis van coördinaten
    double de = WiskundeUtil.berekenAfstandTussen(_d, _e);
    double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
    double df = WiskundeUtil.berekenAfstandTussen(_d, _f);

    //Koppeling zijden aan de letters a, b, c voor cosinusregel
    double a = ef;
    double b = df;
    double c = de;

    double beta1 = WiskundeUtil.berekenHoekBetaVanDriehoek(a, b, c);

    //Berekening lengtes van de zijden van driehoek 2 op basis van coördinaten
    //ef is al berekend
    double fg = WiskundeUtil.berekenAfstandTussen(_f, _g);
    double eg = WiskundeUtil.berekenAfstandTussen(_e, _g);

    //Koppeling zijden aan de letters a, b, c voor cosinusregel
    a = fg;
    b = eg;
    c = ef;

    double alpha2 = WiskundeUtil.berekenHoekAlphaVanDriehoek(a, b, c);

    double hoekE = beta1 + alpha2;
```

```
            return hoekE;
        }

        public double berekenHoekF()
        {
            //Berekening lengtes van de zijden van driehoek 1 op basis van coördinaten
            double de = WiskundeUtil.berekenAfstandTussen(_d, _e);
            double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
            double df = WiskundeUtil.berekenAfstandTussen(_d, _f);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            double a = ef;
            double b = df;
            double c = de;

            double gamma1 = WiskundeUtil.berekenHoekGammaVanDriehoek(a, b, c);

            //Berekening lengtes van de zijden van driehoek 2 op basis van coördinaten
            //ef is al berekend
            double fg = WiskundeUtil.berekenAfstandTussen(_f, _g);
            double eg = WiskundeUtil.berekenAfstandTussen(_e, _g);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            a = fg;
            b = eg;
            c = ef;

            double beta2 = WiskundeUtil.berekenHoekBetaVanDriehoek(a, b, c);

            double hoekF = gamma1 + beta2;

            return hoekF;
        }

        public double berekenHoekG()
        {
            //Zijden van driehoek 1 zijn n.v.t. voor het berekenen van hoek G

            //Berekening lengtes van de zijden van driehoek 2 op basis van coördinaten
            double ef = WiskundeUtil.berekenAfstandTussen(_e, _f);
            double fg = WiskundeUtil.berekenAfstandTussen(_f, _g);
            double eg = WiskundeUtil.berekenAfstandTussen(_e, _g);

            //Koppeling zijden aan de letters a, b, c voor cosinusregel
            double a = fg;
            double b = eg;
            double c = ef;

            double gamma2 = WiskundeUtil.berekenHoekGammaVanDriehoek(a, b, c);

            double hoekG = gamma2;

            return hoekG;
        }
    }
}
```

Automatische test (Versie A)

HoekTest.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Text;
using System.Collections.Generic;
using InfoSupport.Experiment.Hoekberekening;
namespace InfoSupport.Experiment.HoekberekeningTest
{
    [TestClass()]
    public class HoekTest
    {
        private TestContext testContextInstance;

        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }

        [TestMethod()]
        public void berekenHoekATest()
        {
            Coordinaat A = new Coordinaat();
            A.x = -1;
            A.y = 3;

            Coordinaat B = new Coordinaat();
            B.x = 3;
            B.y = 0;

            Coordinaat C = new Coordinaat();
            C.x = -1;
            C.y = -1;

            IDriehoek driehoek = new Veelhoek();

            driehoek.A = A;
            driehoek.B = B;
            driehoek.C = C;

            double verwacht = 53.1;
            double daadwerkelijk = Math.Round(driehoek.berekenHoekA(),1);

            Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
        }

        [TestMethod()]
        public void berekenHoekBTest()
        {
            Coordinaat A = new Coordinaat();
            A.x = -1;
            A.y = 3;

            Coordinaat B = new Coordinaat();
            B.x = 3;
            B.y = 0;

            Coordinaat C = new Coordinaat();
            C.x = -1;
            C.y = -1;
```

```
        IDriehoek driehoek = new Veelhoek();

        driehoek.A = A;
        driehoek.B = B;
        driehoek.C = C;

        double verwacht = 50.9;
        double daadwerkelijk = Math.Round(driehoek.berekenHoekB(), 1);

        Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
    }

    [TestMethod()]
    public void berekenHoekCTest()
    {
        Coordinaat A = new Coordinaat();
        A.x = -1;
        A.y = 3;

        Coordinaat B = new Coordinaat();
        B.x = 3;
        B.y = 0;

        Coordinaat C = new Coordinaat();
        C.x = -1;
        C.y = -1;

        IDriehoek driehoek = new Veelhoek();

        driehoek.A = A;
        driehoek.B = B;
        driehoek.C = C;

        double verwacht = 76.0;
        double daadwerkelijk = Math.Round(driehoek.berekenHoekC(), 1);

        Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
    }

    [TestMethod()]
    public void berekenHoekDTest()
    {
        Coordinaat D = new Coordinaat();
        D.x = -1;
        D.y = 3;

        Coordinaat E = new Coordinaat();
        E.x = 3;
        E.y = 0;

        Coordinaat F = new Coordinaat();
        F.x = -1;
        F.y = -1;

        Coordinaat G = new Coordinaat();
        G.x = 5;
        G.y = -3;

        IVierhoek vierhoek = new Veelhoek();

        vierhoek.D = D;
        vierhoek.E = E;
        vierhoek.F = F;
        vierhoek.G = G;

        double verwacht = 53.1;
        double daadwerkelijk = Math.Round(vierhoek.berekenHoekD(), 1);

        Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
    }
```

```
[TestMethod()]
public void berekenHoekETest()
{
    Coordinaat D = new Coordinaat();
    D.x = -1;
    D.y = 3;

    Coordinaat E = new Coordinaat();
    E.x = 3;
    E.y = 0;

    Coordinaat F = new Coordinaat();
    F.x = -1;
    F.y = -1;

    Coordinaat G = new Coordinaat();
    G.x = 5;
    G.y = -3;

    IVierhoek vierhoek = new Veelhoek();

    vierhoek.D = D;
    vierhoek.E = E;
    vierhoek.F = F;
    vierhoek.G = G;

    double verwacht = 160.6;
    double daadwerkelijk = Math.Round(vierhoek.berekenHoekE(), 1);

    Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
}

[TestMethod()]
public void berekenHoekFTest()
{
    Coordinaat D = new Coordinaat();
    D.x = -1;
    D.y = 3;

    Coordinaat E = new Coordinaat();
    E.x = 3;
    E.y = 0;

    Coordinaat F = new Coordinaat();
    F.x = -1;
    F.y = -1;

    Coordinaat G = new Coordinaat();
    G.x = 5;
    G.y = -3;

    IVierhoek vierhoek = new Veelhoek();

    vierhoek.D = D;
    vierhoek.E = E;
    vierhoek.F = F;
    vierhoek.G = G;

    double verwacht = 108.4;
    double daadwerkelijk = Math.Round(vierhoek.berekenHoekF(), 1);

    Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
}

[TestMethod()]
public void berekenHoekGTest()
{
    Coordinaat D = new Coordinaat();
    D.x = -1;
```

```
            D.y = 3;

            Coordinaat E = new Coordinaat();
            E.x = 3;
            E.y = 0;

            Coordinaat F = new Coordinaat();
            F.x = -1;
            F.y = -1;

            Coordinaat G = new Coordinaat();
            G.x = 5;
            G.y = -3;

            IVierhoek vierhoek = new Veelhoek();

            vierhoek.D = D;
            vierhoek.E = E;
            vierhoek.F = F;
            vierhoek.G = G;

            double verwacht = 37.9;
            double daadwerkelijk = Math.Round(vierhoek.berekenHoekG(), 1);

            Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
        }

        [TestMethod()]
        public void berekenHoekJTest()
        {
            Coordinaat H = new Coordinaat();
            H.x = -1;
            H.y = 3;

            Coordinaat I = new Coordinaat();
            I.x = 3;
            I.y = 0;

            Coordinaat J = new Coordinaat();
            J.x = -1;
            J.y = -1;

            Coordinaat K = new Coordinaat();
            K.x = 5;
            K.y = -3;

            Coordinaat L = new Coordinaat();
            L.x = -5;
            L.y = -6;

            //TODO: Uncomment onderstaande code en
            //      instantieer object van type IVijfhoek

            //IVijfhoek vijkhoek = new ...

            //vijfhoek.H = H;
            //vijfhoek.I = I;
            //vijfhoek.J = J;
            //vijfhoek.K = K;
            //vijfhoek.L = L;

            //double verwacht = 218.7;
            //double daadwerkelijk = Math.Round(vijfhoek.berekenHoekJ(), 1);

            //Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
        }
    }
}
```

Automatische test (Versie B)

HoekTest.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Text;
using System.Collections.Generic;
using InfoSupport.Experiment.Hoekberekening;
namespace InfoSupport.Experiment.HoekberekeningTest
{
    [TestClass()]
    public class HoekTest
    {
        private TestContext testContextInstance;

        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }

        [TestMethod()]
        public void berekenHoekATest()
        {
            Coordinaat A = new Coordinaat();
            A.x = -1;
            A.y = 3;

            Coordinaat B = new Coordinaat();
            B.x = 3;
            B.y = 0;

            Coordinaat C = new Coordinaat();
            C.x = -1;
            C.y = -1;

            IDriehoek driehoek = new Driehoek();

            driehoek.A = A;
            driehoek.B = B;
            driehoek.C = C;

            double verwacht = 53.1;
            double daadwerkelijk = Math.Round(driehoek.berekenHoekA(),1);

            Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
        }

        [TestMethod()]
        public void berekenHoekBTest()
        {
            Coordinaat A = new Coordinaat();
            A.x = -1;
            A.y = 3;

            Coordinaat B = new Coordinaat();
            B.x = 3;
            B.y = 0;

            Coordinaat C = new Coordinaat();
            C.x = -1;
            C.y = -1;
```

```
        IDriehoek driehoek = new Driehoek();

        driehoek.A = A;
        driehoek.B = B;
        driehoek.C = C;

        double verwacht = 50.9;
        double daadwerkelijk = Math.Round(driehoek.berekenHoekB(), 1);

        Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
    }

    [TestMethod()]
    public void berekenHoekCTest()
    {
        Coordinaat A = new Coordinaat();
        A.x = -1;
        A.y = 3;

        Coordinaat B = new Coordinaat();
        B.x = 3;
        B.y = 0;

        Coordinaat C = new Coordinaat();
        C.x = -1;
        C.y = -1;

        IDriehoek driehoek = new Driehoek();

        driehoek.A = A;
        driehoek.B = B;
        driehoek.C = C;

        double verwacht = 76.0;
        double daadwerkelijk = Math.Round(driehoek.berekenHoekC(), 1);

        Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
    }

    [TestMethod()]
    public void berekenHoekDTest()
    {
        Coordinaat D = new Coordinaat();
        D.x = -1;
        D.y = 3;

        Coordinaat E = new Coordinaat();
        E.x = 3;
        E.y = 0;

        Coordinaat F = new Coordinaat();
        F.x = -1;
        F.y = -1;

        Coordinaat G = new Coordinaat();
        G.x = 5;
        G.y = -3;

        IVierhoek vierhoek = new Vierhoek();

        vierhoek.D = D;
        vierhoek.E = E;
        vierhoek.F = F;
        vierhoek.G = G;

        double verwacht = 53.1;
        double daadwerkelijk = Math.Round(vierhoek.berekenHoekD(), 1);

        Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
    }
```

```
[TestMethod()]
public void berekenHoekETest()
{
    Coordinaat D = new Coordinaat();
    D.x = -1;
    D.y = 3;

    Coordinaat E = new Coordinaat();
    E.x = 3;
    E.y = 0;

    Coordinaat F = new Coordinaat();
    F.x = -1;
    F.y = -1;

    Coordinaat G = new Coordinaat();
    G.x = 5;
    G.y = -3;

    IVierhoek vierhoek = new Vierhoek();

    vierhoek.D = D;
    vierhoek.E = E;
    vierhoek.F = F;
    vierhoek.G = G;

    double verwacht = 160.6;
    double daadwerkelijk = Math.Round(vierhoek.berekenHoekE(), 1);

    Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
}

[TestMethod()]
public void berekenHoekFTest()
{
    Coordinaat D = new Coordinaat();
    D.x = -1;
    D.y = 3;

    Coordinaat E = new Coordinaat();
    E.x = 3;
    E.y = 0;

    Coordinaat F = new Coordinaat();
    F.x = -1;
    F.y = -1;

    Coordinaat G = new Coordinaat();
    G.x = 5;
    G.y = -3;

    IVierhoek vierhoek = new Vierhoek();

    vierhoek.D = D;
    vierhoek.E = E;
    vierhoek.F = F;
    vierhoek.G = G;

    double verwacht = 108.4;
    double daadwerkelijk = Math.Round(vierhoek.berekenHoekF(), 1);

    Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
}

[TestMethod()]
public void berekenHoekGTest()
{
    Coordinaat D = new Coordinaat();
    D.x = -1;
```

```
            D.y = 3;

            Coordinaat E = new Coordinaat();
            E.x = 3;
            E.y = 0;

            Coordinaat F = new Coordinaat();
            F.x = -1;
            F.y = -1;

            Coordinaat G = new Coordinaat();
            G.x = 5;
            G.y = -3;

            IVierhoek vierhoek = new Vierhoek();

            vierhoek.D = D;
            vierhoek.E = E;
            vierhoek.F = F;
            vierhoek.G = G;

            double verwacht = 37.9;
            double daadwerkelijk = Math.Round(vierhoek.berekenHoekG(), 1);

            Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
        }

        [TestMethod()]
        public void berekenHoekJTest()
        {
            Coordinaat H = new Coordinaat();
            H.x = -1;
            H.y = 3;

            Coordinaat I = new Coordinaat();
            I.x = 3;
            I.y = 0;

            Coordinaat J = new Coordinaat();
            J.x = -1;
            J.y = -1;

            Coordinaat K = new Coordinaat();
            K.x = 5;
            K.y = -3;

            Coordinaat L = new Coordinaat();
            L.x = -5;
            L.y = -6;

            //TODO: Uncomment onderstaande code en
            //      instantieer object van type IVijfhoek

            //IVijfhoek vijkhoek = new ...

            //vijfhoek.H = H;
            //vijfhoek.I = I;
            //vijfhoek.J = J;
            //vijfhoek.K = K;
            //vijfhoek.L = L;

            //double verwacht = 218.7;
            //double daadwerkelijk = Math.Round(vijfhoek.berekenHoekJ(), 1);

            //Assert.AreEqual(verwacht, daadwerkelijk, "Hoeken ongelijk");
        }
    }
}
```

I. Broncode applicatie 3 : Belasting

Versie A

TrekkerOplegger.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Belastingen
{
    public class TrekkerOplegger
    {
        private int _trekkerGewicht;
        private double _trekkerKostPrijs;
        private int _trekkerBouwJaar;
        private int _opleggerGewicht;
        private double _opleggerKostPrijs;
        private int _opleggerBouwJaar;

        public TrekkerOplegger(int trekkerGewicht, double trekkerKostPrijs, int trekkerBouwJaar,
                               int opleggerGewicht, double opleggerKostPrijs, int
opleggerBouwJaar)
        {
            TrekkerGewicht = trekkerGewicht;
            TrekkerKostPrijs = trekkerKostPrijs;
            TrekkerBouwJaar = trekkerBouwJaar;
            OpleggerGewicht = opleggerGewicht;
            OpleggerKostPrijs = opleggerKostPrijs;
            OpleggerBouwJaar = opleggerBouwJaar;
        }

        public int TrekkerGewicht
        {
            set
            {
                _trekkerGewicht = value;
            }
        }

        public double TrekkerKostPrijs
        {
            set
            {
                _trekkerKostPrijs = value;
            }
        }

        public int TrekkerBouwJaar
        {
            set
            {
                _trekkerBouwJaar = value;
            }
        }

        public int OpleggerGewicht
        {
            set
            {
                _opleggerGewicht = value;
            }
        }

        public double OpleggerKostPrijs
        {
```

```
            set
            {
                _opleggerKostPrijs = value;
            }
        }

        public int OpleggerBouwJaar
        {
            set
            {
                _opleggerBouwJaar = value;
            }
        }

        public int TrekkerOpleggerGewicht
        {
            get
            {
                return (_trekkerGewicht + _opleggerGewicht);
            }
        }

        public double berekenBelasting()
        {
            int huidigJaar = 2007;
            double waardevermindering;
            double trekkerBelasting, opleggerBelasting;

            //berekening van de belasting voor de trekker
            waardevermindering = _trekkerKostPrijs * (huidigJaar - _trekkerBouwJaar) / 10;

            if (_trekkerGewicht <= 1500) {
                    trekkerBelasting = ((_trekkerKostPrijs - waardevermindering)/75)*1.1;
            }
            else if (_trekkerGewicht <= 2250) {
                     trekkerBelasting = ((_trekkerKostPrijs - waardevermindering)/75)*1.2;
            }
            else
                    trekkerBelasting = ((_trekkerKostPrijs - waardevermindering)/75)*1.3;

            // berkening van de belasting voor de oplegger
            waardevermindering = _opleggerKostPrijs * (huidigJaar - _opleggerBouwJaar)/10;
            if (_opleggerGewicht <= 1500) {
                    opleggerBelasting = ((_opleggerKostPrijs - waardevermindering)/85)*1.1;
            }
            else if (_opleggerGewicht <= 2250) {
                     opleggerBelasting = ((_opleggerKostPrijs - waardevermindering)/85)*1.2;
            }
            else
                    opleggerBelasting = ((_opleggerKostPrijs - waardevermindering)/85)*1.3;

            // geef de som van beide belastingen terug
            return (trekkerBelasting + opleggerBelasting);
        }

        public void toonGegevens()
        {
            Console.Out.Write(
                    "De eigenschappen van de trekker-oplegger zijn als volgt:" + "\n" +
                    "Het gewicht van de trekker is: " + _trekkerGewicht  + "\n" +
                    "De kosten van de trekker zijn: " + _trekkerKostPrijs + "\n" +
                    "Het bouwjaar van de trekker is: " + _trekkerBouwJaar  + "\n" +
                    "Het gewicht van de oplegger is: " + _opleggerGewicht + "\n" +
                "De kosten van de oplegger zijn: " + _opleggerKostPrijs + "\n" +
                "Het bouwjaar van de oplegger is: " + _opleggerBouwJaar + "\n" +
                "De belasting voor de trekker-oplegger is: " + berekenBelasting() + "\n"
            );
        }
    }
}
```

Versie B

Oplegger.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Belastingen
{
    public class Oplegger
    {
        private int _gewicht;
        private double _kostPrijs;
        private int _bouwJaar;

        public Oplegger(int gewicht, double kostPrijs, int bouwjaar)
        {
            Gewicht = gewicht;
               KostPrijs = kostPrijs;
               BouwJaar = bouwjaar;
        }

        public int Gewicht
        {
            get
            {
                return _gewicht;
            }
            set
            {
                _gewicht = value;
            }

        }

        public double KostPrijs
        {
            set
            {
                _kostPrijs = value;
            }
        }

        public int BouwJaar
        {
            set
            {
                _bouwJaar = value;
            }
        }

        public double berekenBelasting()
        {
            int huidigJaar = 2007;
            double waardevermindering;
            double belasting;

            waardevermindering = _kostPrijs * (huidigJaar - _bouwJaar)/10;
            if (_gewicht <= 1500) {
                   belasting = ((_kostPrijs - waardevermindering)/85) * 1.1;
            }
            else if (_gewicht <= 2250) {
                   belasting = ((_kostPrijs - waardevermindering)/85) * 1.2;
            }
            else
                   belasting = ((_kostPrijs - waardevermindering)/85) * 1.3;

            return belasting;
        }
```

```
        public void toonGegevens()
        {
            Console.Out.Write(
               "De eigenschappen van de oplegger zijn als volgt:" + "\n" +
               "Het gewicht is: " + Gewicht + "\n" +
               "De kosten zijn: " + _kostPrijs + "\n" +
               "Het bouwjaar is: " + _bouwJaar + "\n" +
               "De belasting voor de oplegger is: " + berekenBelasting() + "\n"
            );
        }
    }
}
```

Trekker.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Belastingen
{
    public class Trekker
    {
        int _gewicht;
            double _kostPrijs;
            int _bouwJaar;

        public Trekker(int gewicht, double kostPrijs, int bouwJaar)
        {
            Gewicht = gewicht;
                KostPrijs = kostPrijs;
                BouwJaar = bouwJaar;
        }

        public int Gewicht
        {
            get
            {
                return _gewicht;
            }
            set
            {
                _gewicht = value;
            }

        }

        public double KostPrijs
        {
            set
            {
                _kostPrijs = value;
            }

        }

        public int BouwJaar
        {
            set
            {
                _bouwJaar = value;
            }
        }

        public double berekenBelasting()
        {
            int huidigJaar = 2007;
            double waardevermindering;
            double belasting;
```

```
            waardevermindering = _kostPrijs * (huidigJaar - _bouwJaar)/10;
            if (_gewicht <= 1500) {
                    belasting = ((_kostPrijs - waardevermindering)/75)*1.1;
            }
            else if (_gewicht <= 2250) {
                    belasting = ((_kostPrijs - waardevermindering)/75)*1.2;
            }
            else
                    belasting = ((_kostPrijs - waardevermindering)/75)*1.3;

            return belasting;
        }

        public void toonGegevens()
        {
            Console.Out.Write(
               "De eigenschappen van de trekker zijn als volgt:" + "\n" +
               "Het gewicht is: " + Gewicht + "\n" +
               "De kosten zijn: " + _kostPrijs + "\n" +
               "Het bouwjaar is: " + _bouwJaar + "\n" +
               "De belasting voor de trekker is: " + berekenBelasting() + "\n"
            );
        }
    }
}
```

TrekkerOplegger.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace InfoSupport.Experiment.Belastingen
{
    public class TrekkerOplegger
    {
        private Trekker trekker;
        private Oplegger oplegger;

        public TrekkerOplegger(Trekker t, Oplegger o)
        {
            trekker = t;
            oplegger = o;
        }

        public int Gewicht
        {
            get
            {
                return trekker.Gewicht + oplegger.Gewicht;
            }
        }

        public double berekenBelasting()
        {
            return trekker.berekenBelasting() + oplegger.berekenBelasting();
        }
    }
}
```

Broncode automatische test (versie A)

BelastingTest.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Text;
using System.Collections.Generic;
using InfoSupport.Experiment.Belastingen;
namespace BelastingTest
{
    [TestClass()]
    public class BelastingTest
    {
        private TestContext testContextInstance;

        double verwacht;
        double daadwerkelijk;

        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }

        [TestMethod()]
        public void berekenBelastingTest1()
        {
            TrekkerOplegger target = new TrekkerOplegger(1750, 15000.0, 2002, 1000, 10000.0,
2003);

            // TODO: Oplegger heeft 8 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in zone 3

            verwacht = 221.68;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }

        [TestMethod()]
        public void berekenBelastingTest2()
        {
            TrekkerOplegger target = new TrekkerOplegger(1450, 20000.0, 2004, 1300, 12500.0,
2004);

            // TODO: Oplegger heeft 12 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in zone 2

            verwacht = 356.28;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }

        [TestMethod()]
        public void berekenBelastingTest3()
        {
            TrekkerOplegger target = new TrekkerOplegger(2300, 18000.0, 2002, 1800, 9000.0,
2004);

            // TODO: Oplegger heeft 4 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in zone 1
```

```
            verwacht = 257.19;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }

        [TestMethod()]
        public void berekenBelastingTest4()
        {
            TrekkerOplegger target = new TrekkerOplegger(2300, 18000.0, 2002, 1800, 9000.0,
2004);

            // TODO: Oplegger heeft 4 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in
            //       andere zone dan zone 1, 2 of 3

            verwacht = 274.33;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }
    }
}
```

Broncode automatische test (versie B)

BelastingTest.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Text;
using System.Collections.Generic;
using InfoSupport.Experiment.Belastingen;
namespace BelastingTest
{
    [TestClass()]
    public class BelastingTest
    {
        private TestContext testContextInstance;

        double verwacht;
        double daadwerkelijk;

        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }

        [TestMethod()]
        public void berekenBelastingTest1()
        {
            Trekker t = new Trekker(1750, 15000.0, 2002);
            Oplegger o = new Oplegger(1000, 10000.0, 2003);
            TrekkerOplegger target = new TrekkerOplegger(t, o);

            // TODO: Oplegger heeft 8 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in zone 3

            verwacht = 221.68;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }
```

```
        [TestMethod()]
        public void berekenBelastingTest2()
        {
            Trekker t = new Trekker(1450, 20000.0, 2004);
            Oplegger o = new Oplegger(1300, 12500.0, 2004);
            TrekkerOplegger target = new TrekkerOplegger(t, o);

            // TODO: Oplegger heeft 12 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in zone 2

            verwacht = 356.28;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }

        [TestMethod()]
        public void berekenBelastingTest3()
        {
            Trekker t = new Trekker(2300, 18000.0, 2002);
            Oplegger o = new Oplegger(1800, 9000.0, 2004);
            TrekkerOplegger target = new TrekkerOplegger(t,o);

            // TODO: Oplegger heeft 4 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in zone 1

            verwacht = 257.19;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }

        [TestMethod()]
        public void berekenBelastingTest4()
        {
            Trekker t = new Trekker(2300, 18000.0, 2002);
            Oplegger o = new Oplegger(1800, 9000.0, 2004);
            TrekkerOplegger target = new TrekkerOplegger(t, o);

            // TODO: Oplegger heeft 4 wielen
            // TODO: De combinatie Trekker-oplegger is geregistreerd in
            //       andere zone dan zone 1, 2 of 3

            verwacht = 274.33;
            daadwerkelijk = Math.Round(target.berekenBelasting(), 2);

            Assert.AreEqual(verwacht, daadwerkelijk, "Andere belasting anders verwacht");
        }
    }
}
```

J. Berekening metrieken applicatie 1: Levensverzekering

Berekening Response For A Class (RFC)

Versie A

Klasse Calculator

Methodendefinities / methodeaanroepen
`public Calculator()`
`initialiseerSterfteKansTabel()`
`public double berekenPremiePerMaand(Persoon persoon, Levensverzekering verzekering)`
`berekenBetaalBedragZonderWinst(sterftekans, verzekering.VerzekerdeBedrag)`
`berekenBedragPerMaand(betaalBedragMetWinst, verzekering.Looptijd)`
`persoon.IsBestaandeKlant`
Totaal: 6

Versie B

Klasse Calculator

Methodendefinities / methodeaanroepen
`public double berekenPremiePerMaand(Persoon persoon, Levensverzekering verzekering)`
`private double berekenBetaalBedragZonderWinst(double sterftekans, double verzekerdBedrag)`
`verzekering.Looptijd`[4]
`persoon.Leeftijd`
`persoon.IsBestaandeKlant`
Totaal: 5

Berekening Coupling Between Objects (CBO)

Versie A

Klasse Calculator

Aantal	Koppeling
1x	`LeeftijdKlasse.Einde`
1x	`LeeftijdKlasse.Junior`
1x	`LeeftijdKlasse.Medior`
1x	`LeeftijdKlasse.Senior`
2x	`LooptijdKlasse.Kort`
2x	`LooptijdKlasse.Gemiddeld`
2x	`LooptijdKlasse.Lang`

[4] Een get-accessor in C# wordt gezien als een methode

1x	persoon.Leeftijd
1x	verzekering.Looptijd
1x	verzekering.VerzekerdeBedrag
1x	Constanten.WINSTFACTOR
1x	verzekering.Looptijd
1x	persoon.IsBestaandeKlant
1x	Constanten.KORTINGSFACTOR
1x	Constanten.GEMIDDELDE_LOOPTIJD_KORT
1x	Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD
1x	Constanten.GEMIDDELDE_LOOPTIJD_LANG
1x	Constanten.MAANDEN_IN_JAAR
1x	Constanten.GEMIDDELD_PERCENTAGE_VOLBRACHTE_LOOPTIJD_BIJ_OVERLIJDEN
Totaal: 22	

Versie B

Klasse Calculator

Aantal	Koppeling
9x	persoon.Leeftijd
10x	verzekering.Looptijd
4x	LooptijdKlasse.Kort[5]
4x	LooptijdKlasse.Gemiddeld
4x	LooptijdKlasse.Lang
3x	LeeftijdKlasse.Junior
3x	LeeftijdKlasse.Medior
3x	LeeftijdKlasse.Senior
1x	verzekering.VerzekerdeBedrag
1x	Constanten.WINSTFACTOR
1x	Constanten.GEMIDDELDE_LOOPTIJD_KORT
1x	Constanten.GEMIDDELDE_LOOPTIJD_GEMIDDELD
1x	Constanten.MAANDEN_IN_JAAR
1x	persoon.IsBestaandeKlant
1x	Constanten.KORTINGSFACTOR
1x	Constanten.GEMIDDELD_PERCENTAGE_VOLBRACHTE_LOOPTIJD_BIJ_OVERLIJDEN
Totaal: 48	

[5] het type enum wordt voor de berekening gezien als het type klasse. Het opvragen van een enum-waarde geldt dus als een koppeling.

Berekening Bandi-metrieken

Versie A

Klasse Calculator

Attributen

Aantal	Attribuut
1x	double[,][6]

Methoden

Aantal	Methode	Berekening per methode	
1x	Constructor	IL	0
		OAC	0
		IS	0
1x	initialiseerSterfteKansTabel : void	IL	0
		OAC	0
		IS	0
1x	berekenPremiePerMaand(Persoon, Levensverzekering)	IL	2*1 + (6*3 + 6*3) = 38
		OAC	6 + 6 = 12
		IS	2 + 12 = 14
1x	berekenBedragPerMaand(double, LooptijdKlasse[7])	IL	2*1 + (2*3 + 1*3) = 11
		OAC	2 + 1 = 3
		IS	2 + 3 = 5
1x	berekenBetaalBedragZonderWinst(double, double)	IL	2*1 + (2*3 + 2*3) = 14
		OAC	2 + 2 = 4
		IS	2 + 4 = 6
	Totaal	IL	1*38 + 1*11 + 1*14 = 63
		OAC	1*12 + 1*3 + 1*4 = 19
		IS	1*14 + 1*5 + 1*6 = 25

[6] De array wordt geteld als waarde 3, zoals is aangegeven in Tabel 21. Het gaat hierbij echter wel om een tweedimensionale array, waardoor de complexiteit stijgt.

[7] Een enum staat niet opgenomen in Tabel 21 en wordt geteld als waarde 1. Voordat een enum bestond in C# werden constante integers gebruikt. Een integer heeft in Tabel 21 de waarde 1.

Versie B

Klasse Calculator

Attributen

Geen attributen

Methoden

Aantal	Methode	Berekening per methode	
3x	berekenPremiePerMaand() : double	IL	0
		OAC	2
		IS	1 + 2 = 3
3x	berekenBetaalBedragZonderWinst() : double	IL	0
		OAC	2
		IS	1 + 2 = 3
	Totaal	IL	0
		OAC	3*2 + 3*2 = 12
		IS	3*3 + 3*3 = 18

K. Berekening metrieken applicatie 2: Hoekberekening

Berekening Response For A Class (RFC)

Versie A

Klasse Veelhoek

Methodendefinities / methodeaanroepen
`public Coordinaat A { set { _a = value; } }`
`public Coordinaat B { set { _a = value; } }`
`public Coordinaat C { set { _a = value; } }`
`public Coordinaat D { set { _a = value; } }`
`public Coordinaat E { set { _a = value; } }`
`public Coordinaat F { set { _a = value; } }`
`public Coordinaat G { set { _a = value; } }`
`public double berekenHoekA()`
`public double berekenHoekB()`
`public double berekenHoekC()`
`public double berekenHoekD()`
`public double berekenHoekE()`
`public double berekenHoekF()`
`public double berekenHoekG()`
`WiskundeUtil.berekenAfstandTussen`
`WiskundeUtil.berekenHoekAlphaVanDriehoek`
`WiskundeUtil.berekenHoekBetaVanDriehoek`
`WiskundeUtil.berekenHoekGammaVanDriehoek`
Totaal: 18

Versie B

Klasse Driehoek

Methodendefinities / methodeaanroepen
`public Coordinaat A { set { _a = value; } }`
`public Coordinaat B { set { _a = value; } }`
`public Coordinaat C { set { _a = value; } }`
`public double berekenHoekA()`
`public double berekenHoekB()`
`public double berekenHoekC()`
`WiskundeUtil.berekenAfstandTussen`
`WiskundeUtil.berekenHoekAlphaVanDriehoek`
`WiskundeUtil.berekenHoekBetaVanDriehoek`

`WiskundeUtil.berekenHoekGammaVanDriehoek`
Totaal: 10

Klasse Vierhoek

Methodendefinities / methodeaanroepen
`public Coordinaat D { set { _a = value; } }`
`public Coordinaat E { set { _a = value; } }`
`public Coordinaat F { set { _a = value; } }`
`public Coordinaat G { set { _a = value; } }`
`public double berekenHoekD()`
`public double berekenHoekE()`
`public double berekenHoekF()`
`public double berekenHoekG()`
`WiskundeUtil.berekenAfstandTussen`
`WiskundeUtil.berekenHoekAlphaVanDriehoek`
`WiskundeUtil.berekenHoekBetaVanDriehoek`
`WiskundeUtil.berekenHoekGammaVanDriehoek`
Totaal: 12

Klasse Driehoek + Vierhoek

Totaal	**10 + 12 = 22**

Berekening Coupling Between Objects (CBO)

Versie A

Klasse Veelhoek

Aantal	Koppeling
25x	`WiskundeUtil.berekenAfstandTussen`
3x	`WiskundeUtil.berekenHoekAlphaVanDriehoek`
3x	`WiskundeUtil.berekenHoekBetaVanDriehoek`
3x	`WiskundeUtil.berekenHoekGammaVanDriehoek`
Totaal: 34	

Versie B

Klasse Driehoek

Aantal	Koppeling
9x	`WiskundeUtil.berekenAfstandTussen`
1x	`WiskundeUtil.berekenHoekAlphaVanDriehoek`

1x	`WiskundeUtil.berekenHoekBetaVanDriehoek`
1x	`WiskundeUtil.berekenHoekGammaVanDriehoek`
Totaal: 12	

Klasse Vierhoek

Aantal	Koppeling
16x	`WiskundeUtil.berekenAfstandTussen`
2x	`WiskundeUtil.berekenHoekAlphaVanDriehoek`
2x	`WiskundeUtil.berekenHoekBetaVanDriehoek`
2x	`WiskundeUtil.berekenHoekGammaVanDriehoek`
Totaal: 22	

Klasse Driehoek + Vierhoek

Totaal	**12 + 22 = 34**

Berekening Bandi-metrieken

Versie A

Klasse Veelhoek

Attributen

Aantal	Attribuut
7x	Coordinaat

Methoden

Aantal	Methode	Berekening per methode	
7x	Set(Coordinaat)	IL	7*1 + (6*6 + 6*6 + 6*6 + 6*6 + 6*6 + 6*6 + 6*6) = 259
		OAC	6
		IS	1 + 6 = 7
7x	berekenHoekX() : double	IL	7*1 + (2*6 + 2*6 + 2*6 + 2*6 + 2*6 + 2*6 + 2*6) = 91
		OAC	2
		IS	1 + 2 = 3
	Totaal	IL	7*259 + 7*91 = 2450
		OAC	7*6 + 7*2 = 56
		IS	7*7 + 7*3 = 70

Versie B

Klasse Driehoek

Attributen

Aantal	Attribuut
3x	Coordinaat

Methoden

Aantal	Methode	Berekening per methode	
3x	Set(Coordinaat)	IL	3* 1 + (6*6 + 6*6 + 6*6) = 111
		OAC	6
		IS	1 + 6 = 7
3x	berekenHoekX() : double	IL	3*1 + (2*6 + 2*6 + 2*6) = 39
		OAC	2
		IS	1 + 2 = 3
	Totaal	IL	3*111 + 3*39 = 450
		OAC	3*6 + 3*2 = 24
		IS	3*7 + 3*3 = 30

Klasse Vierhoek

Attributen

Aantal	Attribuut
4x	Coordinaat

Methoden

Aantal	Methode	Berekening per methode	
4x	Set(Coordinaat)	IL	4*1 + (6*6 + 6*6 + 6*6 + 6*6) = 148
		OAC	6
		IS	1 + 6 = 7
4x	berekenHoekX() : double	IL	4*1 + (2*6 + 2*6 + 2*6 + 2*6) = 52
		OAC	2
		IS	1 + 2 = 3
	Totaal	IL	4*148 + 4*52 = 800
		OAC	4*6 + 4*2 = 32
		IS	4*7 + 4*3 = 40

Klasse Driehoek + Vierhoek

Totaal

IL	450 + 800 = 1250
OAC	24 + 32 = 56
IS	30 + 40 =70

L. Berekening metrieken applicatie 3: Belasting

Berekening Response For A Class (RFC)

Versie A

Klasse TrekkerOplegger

Methodendefinities / methodeaanroepen
`public TrekkerOplegger()`
`public int TrekkerGewicht{ set{} }`
`public double TrekkerKostPrijs { set{} }`
`public int TrekkerBouwJaar{ set{} }`
`public int OpleggerGewicht{ set{} }`
`public double OpleggerKostPrijs { set{} }`
`public int OpleggerBouwJaar{ set{} }`
`public int TrekkerOpleggerGewicht { get{ }}`
`public double berekenBelasting()`
`public void toonGegevens()`
`Console.Out.Write`
Totaal: 11

Versie B

Klasse Trekker

Methodendefinities / methodeaanroepen
`public Trekker(int gewicht, double kostPrijs, int bouwJaar)`
`public int Gewicht{ get{ return _gewicht; } }`
`public int Gewicht{ set{_gewicht = value; } }`
`public int KostPrijs{ aet{_kostPrijs = value; } }`
`public int BouwJaar{ aet{_bouwJaar = value; } }`
`public double berekenBelasting()`
`public void toonGegevens()`
`Console.Out.Write`
Totaal: 8

Klasse Oplegger

Methodendefinities / methodeaanroepen
`public Oplegger(int gewicht, double kostPrijs, int bouwjaar)`
`public int Gewicht{ get{ return _gewicht; } }`
`public int Gewicht{ set{_gewicht = value; } }`
`public int KostPrijs{ aet{_kostPrijs = value; } }`

`public int BouwJaar{ aet{_bouwJaar = value; } }`
`public double berekenBelasting()`
`public void toonGegevens()`
`Console.Out.Write`
Totaal: 8

Klasse TrekkerOplegger

Methodendefinities / methodeaanroepen
`public TrekkerOplegger(Trekker t, Oplegger o)`
`public int Gewicht { get {return trekker.Gewicht + oplegger.Gewicht; }}`
`trekker.Gewicht`
`oplegger.Gewicht`
`public double berekenBelasting()`
`trekker.berekenBelasting()`
`oplegger.berekenBelasting();`
Totaal: 7

Klasse Trekker + Klasse Oplegger + Klasse TrekkerOplegger

Totaal	**8 + 8 + 7 = 23**

Berekening Coupling Between Objects (CBO)

Versie A

Klasse TrekkerOplegger

Geen koppelingen, dus CBO = 0.

Versie B

Klasse Trekker

Geen koppelingen, dus CBO = 0.

Klasse Oplegger

Geen koppelingen, dus CBO = 0.

Klasse TrekkerOplegger

Aantal	Koppeling
1x	`trekker.Gewicht`
1x	`oplegger.Gewicht`
1x	`trekker.berekenBelasting()`
1x	`Oplegger.berekenBelasting()`
Totaal: 4	

Klasse Trekker + Klasse Oplegger + Klasse TrekkerOplegger

Totaal	**0 + 0 + 4 = 4**

Berekening Bandi-metrieken

Versie A

Klasse TrekkerOplegger

Attributen

Aantal	Attribuut
4x	int
2x	double

Methoden

Aantal	Methode	Berekening per methode	
4x	Set(int)	IL	6*1 + (1*1 + 1*1 + 1*1 + 1*1 + 1*2 + 1*2) = 14
		OAC	1
		IS	1 + 1 = 2
2x	Set(double)	IL	6*1 + (2*1 + 2*1 + 2*1 + 2*1 + 2*2 + 2*2) = 22
		OAC	2
		IS	1 + 2 =3
1x	berekenBelasting() : double	IL	6*1 + (2*1 + 2*1 + 2*1 + 2*1 + 2*2 + 2*2) = 22
		OAC	2
		IS	1 + 2 =3
1x	Get() : int	IL	6*1 + (1*1 + 1*1 + 1*1 + 1*1 + 1*2 + 1*2) = 14
		OAC	1
		IS	1 + 1 = 2
1x	Constructor(4x int, 2x double)	IL	6*6 + (1*1 + 1*1 + 1*1 + 1*1 + 1*2 + 1*2 + 1*1 + 1*1 + 1*1 + 1*1 + 1*2 + 1*2 + 1*1 + 1*1 + 1*1 + 1*1 + 1*2 + 1*2 + 1*1 + 1*1 + 1*1 + 1*1 + 1*2 + 1*2 + 2*1 + 2*1 + 2*1 + 2*1 + 2*2 + 2*2 + 2*1 + 2*1 + 2*1 + 2*1 + 2*2 + 2*2) = 36 + 8 + 8 + 8 + 8 + 16 + 16 = 100
		OAC	1 + 1 + 1 + 1 + 2 + 2 = 8
		IS	6 + 8 = 14
	Totaal	IL	4*14 + 2*22 + 1*22 + 1*14 + 1*100 = 236
		OAC	4*1 + 2*2 + 1*2 + 1*1 + 1*8 = 19
		IS	4*2 + 2*3 + 1*3 + 1*2 + 1*14 = 33

Versie B

Klasse Trekker

Attributen

Aantal	Attribuut
2x	int
1x	double

Methoden

Aantal	Methode	Berekening per methode	
2x	Set(int)	IL	3*1 + (1*1 + 1*1 + 1*2) = 7
		OAC	1
		IS	1 + 1 = 2
1x	Set(double)	IL	3*1 + (2*1 + 2*1 + 2*2) = 11
		OAC	2
		IS	1 + 2 =3
1x	berekenBelasting() : double	IL	3*1 + (2*1 + 2*1 + 2*2) = 11
		OAC	2
		IS	1 + 2 =3
1x	Get() : int	IL	3*1 + (1*1 + 1*1 + 1*2) = 7
		OAC	1
		IS	1 + 1 = 2
1x	Constructor(2x int, 1x double)	IL	3*3 + (1*1 + 1*1 + 1*2 + 1*1 + 1*1 + 1*2 + 2*1 + 2*1 + 2*2) = 9 + 4 + 4 + 8 = 25
		OAC	1 + 1 + 2 = 4
		IS	3 + 4 = 7
	Totaal	IL	2*7 + 1*11 + 1*11 + 1*7 + 1*25 = 68
		OAC	2*1 + 1*2 + 1*2 + 1*1 + 1*4 = 11
		IS	2*2 + 1*3 + 1*3 + 1*2 + 1*7 = 19

Klasse Oplegger

Attributen

Aantal	Attribuut
2x	int
1x	double

Methoden

Aantal	Methode	Berekening per methode	
2x	Set(int)	IL	3*1 + (1*1 + 1*1 + 1*2) = 7
		OAC	1
		IS	1 + 1 = 2
1x	Set(double)	IL	3*1 + (2*1 + 2*1 + 2*2) = 11
		OAC	2
		IS	1 + 2 =3
1x	berekenBelasting() : double	IL	3*1 + (2*1 + 2*1 + 2*2) = 11
		OAC	2
		IS	1 + 2 =3
1x	Get() : int	IL	3*1 + (1*1 + 1*1 + 1*2) = 7
		OAC	1
		IS	1 + 1 = 2
1x	Constructor(2x int, 1x double)	IL	3*3 + (1*1 + 1*1 + 1*2 + 1*1 + 1*1 + 1*2 + 2*1 + 2*1 + 2*2) = 9 + 4 + 4 + 8 = 25
		OAC	1 + 1 + 2 = 4
		IS	3 + 4 = 7
	Totaal	IL	2*7 + 1*11 + 1*11 + 1*7 + 1*25 = 68
		OAC	2*1 + 1*2 + 1*2 + 1*1 + 1*4 = 11
		IS	2*2 + 1*3 + 1*3 + 1*2 + 1*7 = 19

Klasse TrekkerOplegger

Attributen

Aantal	Attribuut
1x	Trekker
1x	Oplegger

Methoden

Aantal	Methode	Berekening per methode	
1x	berekenBelasting() : double	IL	2*1 + (2*6 + 2*6) = 26
		OAC	2
		IS	1 + 2 = 3
1x	Get() : int	IL	2*1 + (1*6 + 1*6) = 14
		OAC	1
		IS	1 + 1 = 2
1x	Constructor(Trekker, Oplegger)	IL	2*2 + (6*6 + 6*6 + 6*6 + 6*6) = 4 + 72 + 72 = 148
		OAC	6 + 6 = 12
		IS	2 + 12 = 14
	Totaal	IL	1*26 + 1*14 + 1*148 = 188
		OAC	1*2 + 1*1 + 1*12 = 15
		IS	1*3 + 1*2 + 1*14 = 19

Klasse Trekker + Klasse Oplegger + Klasse TrekkerOplegger

	Totaal	IL	68 + 68 + 188 = 324
		OAC	11 + 11 + 15 = 37
		IS	19 + 19 + 19 = 57

M. Ingevulde vragenlijst applicatie 1: Levensverzekering

Deelnemer 1

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Bij 1 niet direct gevonden. Zat even te kijken wat ik met die rare tabellen moest doen (sterftekans). Bij 2 methode aanpassen berekenBetaalbedragZonderWinst werkte niet.
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Tabel sterftekans is niet echt een OO manier. Het aanpassen is nu meer hacken dan code onderhouden.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 2

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja, maar de test faalde omdat ik niet wist / was vergeten om de solution na elke aanpassing te rebuilden.
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Het gebruik van "Begin" en "Eind" waardes in een enum ben ik nooit zo tegengekomen en lijkt me onjuist. Ook het initialiseren van arrays hiermee is misschien onjuist
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 3

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee, al heb ik wel een extra double array moeten maken voor nieuwe sterftekans.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 4

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Nee, de toeslag had ik verkeerd begrepen. Berekening had ik gedaan door de toeslag ("prijs zonder winst x 1.1") op te tellen bij de "prijs zonder winst. Dit optellen moest niet. De toeslag was misleidend.
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee, Nee, Nee.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Ja, de toeslag begreep ik verkeerd.

Deelnemer 5

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Nee, ik zat op de goede weg, maar ik had per ongeluk twee if statements verwijderd, die ik later over het hoofd zag.
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Ja, wel onlogische structuren, maar geen invloed op de aanpassingen of analyse.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Stomme fout van mezelf! Zorgde voor lang werk.

Deelnemer 6

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Ja, je kunt ook "switchen" op enums. Heeft verder geen invloed gehad. Tevens veel geneste ifs.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 7

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Ik zou een switch gebruiken voor de if statements. Dit heeft geen invloed gehad.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	N.v.t.

Deelnemer 8

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ik had direct de juiste klasse te pakken. Opdracht ging erg soepel.
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Grote hoeveelheid if/else. De oplossing heb ik op dezelfde manier geïmplementeerd (dus nog meer if/else)
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	-

N. Ingevulde vragenlijst applicatie 2: Hoekberekening

Deelnemer 1

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Berekende J door 360 – γ1 – β2 – α3 te doen. Aangezien J in het plaatje buiten de figuur stond. Beschrijving beter lezen dus…
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	- Onlogische Veelhoek - Geen invloed op analyse, ben dezelfde weg ingeslagen zoals de rest van de applicatie was opgezet - Ja, zou de applicatie anders opzetten om zo eenvoudigere aanpassingen mogelijk maken.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Veelhoek is een vreemde klasse. Die zou ik wel willen refactoren.

Deelnemer 2

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Het maken van de klasse 'Veelhoek' die IDriehoek en IVierhoek implementeert is mijns inziens de omgekeerde wereld. (Beter zou zijn een algemene interface Veelhoek met concrete klassen Driehoek, Vierhoek, etc.)
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	-

Deelnemer 3

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Ja, 1 veelhoek object die zowel IDriehoek als IVierhoek implementeert is niet echt netjes. Wel invloed: heb ook IVijfhoek hier maar geïmplementeerd.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee.

Deelnemer 4

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja had alles in 1 keer goed, alleen een fout met kopiëren en plakken waardoor de optelling 2 + 2 + 3 was van de hoeken. Dit moest zijn: 1 + 2 + 3. Voor de rest was het goed.
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Nee Nee Nee
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Ja, goed variabelenamen controleren. Eigenfout.

Deelnemer 5

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 6

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 7

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Nee, eerst even zoeken. Zijden verkeerd om ingevuld in formules voor zijdeberekening (verkeerd gelezen).
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee.

Deelnemer 8

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Nee, code was zeer logisch opgebouwd
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee. De code was erg logisch opgebouwd. De aanpassing is ook in de lijn van de applicatie geïmplementeerd. Door de logische structuur kon de aanpassing snel gemaakt worden.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	-

O. Ingevulde vragenlijst applicatie 3: Belasting

Deelnemer 1

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Direct goede oplossing
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Lange methoden, geen hulp methoden. Geen invloed
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Voor de leesbaarheid gebruik maken van hulpmethoden.

Deelnemer 2

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Een klasse "TrekkerOplegger" is niet echt logisch, beter scheiden. Bovendien zonder de public Properties geen setters maar getters moeten zijn. De belasting kan in andere klasse worden berekend.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	-

Deelnemer 3

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja, alleen een fout gemaakt in de if-conditie dus het duurde wat langer.
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Ja, alles in 1 klase is niet erg netjes. Aanpassingen zijn hierdoor ook wat ranzig gehackt.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 4

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja, op een aantal declaraties van variabelen na had ik alles wel in 1 keer.
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Ja onlogische declaraties van meer dezelfde variabelen. Nee Ja, aanpassing lokale variabelen (deze had ik eerst verkeerd staan
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 5

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 6

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Ja
Heb je onlogische structuren in de code ontdekt? Zo ja, heeft dit invloed gehad op de analyse? Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?	Nee
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 7

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Nee, oplegger toevoeging niet opgeteld bij oude belasting
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Constructor variabelenamen met een letter ipv woord. Geen invloed.
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	Nee

Deelnemer 8

Had je direct de juiste oplossing gevonden? Zo nee, beschrijf dan ook welke "verkeerde" weg je was ingeslagen.	Geen verkeerde weg. Duurde wel even voordat ik de juiste functies had gevonden.
Heb je onlogische structuren in de code ontdekt? **Zo ja, heeft dit invloed gehad op de analyse?** **Zo ja, heeft dit invloed gehad op de gemaakte aanpassingen?**	Ja, rare if/else statements met en zonder { }. Ik moest even goed kijken wat daar gebeurde. Geen invloed op aanpassing
Heb je nog aanvullende op- of aanmerkingen over de applicatie en/of jouw gemaakte aanpassingen?	-

P. Instructie voor de leesproef

Situatie

Je bent deelnemer aan een experiment, waarin aanpassingen op drie kleine applicaties uitgevoerd moeten worden. Het gaat om de volgende applicaties:

- Voorbeeldapplicatie 1 : Levensverzekering
- Voorbeeldapplicatie 2: Hoekberekening
- Voorbeeldapplicatie 3 : Belasting

Per applicatie is een aparte omschrijving van het probleem en opdrachtformulering gemaakt, welke op papier beschikbaar worden gesteld.

Voorbereiding

Zorg ervoor dat:

- de opdrachtomschrijving van de drie applicaties en deze instructie op papier uitgeprint zijn.
- je een klok in de buurt hebt.
- je een pen of potlood bij de hand hebt om de tijd te kunnen registreren.
- je op een rustige locatie bevindt waarin je niet gestoord kunt worden.

Algemene opmerking

Tijdens het oorspronkelijk experiment is de volgende opmerking gemaakt, dus deze wordt nu ook aan jou medegedeeld:

“In Voorbeeldapplicatie 1 : Levensverzekering wordt een tabel gepresenteerd met de nieuwe sterftekans voor rokers. Het is niet zo dat deze rokerstoeslag op de een of andere manier in de tabel verwerkt is. De later beschreven rokerstoeslag staat dus compleet los van deze tabel. ”.

Lezen

Voer voor elke opdracht onderstaande stappen uit. Neem een korte pauze tussen de verschillende applicaties.

1. Noteer de begintijd in Tabel 57 (op de minuut nauwkeurig).
2. **Lees de opdrachtomschrijving van de applicatie op zo’n manier door zodat je er vertrouwen in hebt de opdracht uit te kunnen voeren.**
3. Noteer de eindtijd in Tabel 57 (op de minuut nauwkeurig).

	Begintijd	Eindtijd
Voorbeeldapplicatie 1 : Levensverzekering		
Voorbeeldapplicatie 2: Hoekberekening		
Voorbeeldapplicatie 3 : Belasting		

Tabel 57 Schema voor registreren van de tijd per applicatie