

# Automatisering

Paul Sohier 0806122  
Sebastiaan Polderman 0820738

31 oktober 2010

## Inhoudsopgave

1	<b>Week 1</b>	<b>3</b>
1.1	<i>Opdrachten bij hoofdstuk 1</i>	3
1.1.1	Opdracht 1	3
1.1.2	Opdracht 2	3
1.1.3	Opdracht 3	3
1.1.4	Opdracht 4	3
1.1.5	Opdracht 5	3
1.1.6	Opdracht 6	4
1.1.7	Opdracht 7	4
1.1.8	Opdracht 8	4
1.1.9	Opdracht 9	4
1.1.10	Opdracht 10	4
2	<b>Week 2</b>	<b>4</b>
2.1	<i>Opdrachten bij hoofdstuk 3</i>	4
2.1.1	Opdracht 1	4
2.1.2	Opdracht 2	4
2.1.3	Opdracht 3	5
2.1.4	Opdracht 5	5
3	<b>Week 3</b>	<b>5</b>
3.1	<i>Opdrachten bij hoofdstuk 4</i>	5
3.1.1	Opdracht 1	5
3.1.2	Opdracht 2	5
3.1.3	Opdracht 3	5
3.1.4	Opdracht 4	6
3.1.5	Opdracht 5	6
3.1.6	Opdracht 6	6
3.1.7	Opdracht 7	6
3.1.8	Opdracht 8	7
4	<b>Week 4</b>	<b>8</b>
4.1	<i>Opdrachten bij hoofdstuk 5</i>	8
4.1.1	Opdracht 1	8
4.1.2	Opdracht 2	9
4.1.3	Opdracht 3	9
4.1.4	Opdracht 4	9
5	<b>Week 5</b>	<b>9</b>
5.1	<i>Opdrachten bij hoofdstuk 7</i>	9

5.1.1	Opdracht 1	9
5.1.2	Opdracht 2	11

# 1 Week 1

## 1.1 Opdrachten bij hoofdstuk 1

1.1.1 Geef voorbeelden van computerprogramma's die voornamelijk ontwikkeld worden volgens het watervalmodel

Een pakket als MS Office zou volgens dit model ontwikkeld worden omdat het van te voren bepaalde eisen heeft aan de functionaliteit. Zodra het basis pakket ontwikkeld is zal er niet veel meer veranderd worden aan het uiteindelijke doel van het programma.

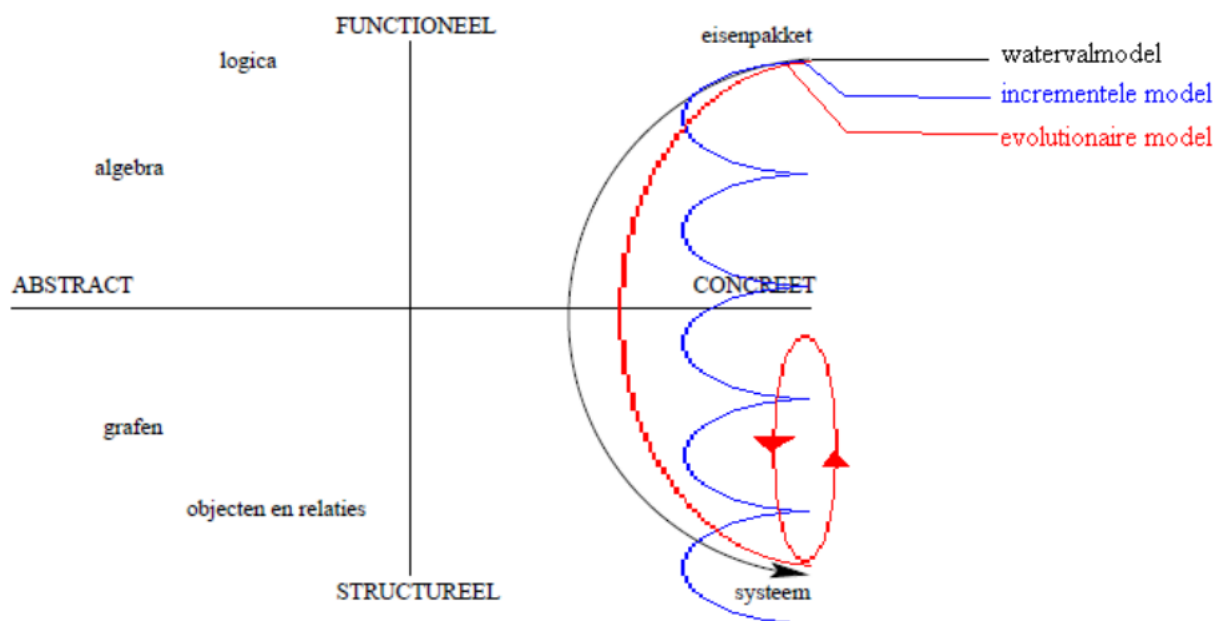
1.1.2 Geef voorbeelden van computerprogramma's die voornamelijk ontwikkeld worden met prototyping.

Een goed voorbeeld hierbij is een project op school. Hierbij wordt eigenlijk bijna direct begonnen met het werken aan het eindproduct, terwijl er nog niet echt bedacht is wat het eindproduct moet worden. Op diverse punten tijdens de ontwikkeling van het programma zal het eisenpakket worden aangepast naar de uiteindelijke wensen.

1.1.3 Wat is het verschil tussen validatie en verificatie

Bij validatie (Valideren) wordt gekeken of het ontwerp voldoet aan het eisenpakket, terwijl er bij verificatie wordt gekeken of het process voldeed.

1.1.4 Hoe zou u de ontwikkeltrajecten van het waterval-, het incrementele- en het evolutionaire model aangeven in figuur 1.1?



1.1.5 Zal de evolutionaire ontwikkelstrategie de grens tussen ontwikkeling en onderhoud laten verdwijnen?

Ja, zodra je onderhoud doet kan je tegelijkertijd ook nieuwe, door de klant gewenste, features kunnen toevoegen aan het al bestaande programma.

#### 1.1.6 Zoek op internet voorbeelde van repositories.

Een voorbeeld hiervan zijn de door debian gebruikte repositories, waaruit alle Debian installaties software vandaan installeren.

#### 1.1.7 Geef de voor en nadelen van open-source software.

Een groot voordeel is dat de sourcecode van de applicatie/programma vrijelijk te bekijken is. Hierdoor zie je dat bijvoorbeeld security problemen welke in deze source code aanwezig zijn sneller gevonden worden en hierdoor het algemener bekend is of software veilig is of niet.

Helaas is dit grote voordeel ook direct een nadeel. Wanneer een security probleem gevonden wordt in een applicatie/programma en dit wordt niet opgelost door de vendor kan dit makkelijk misbruikt worden door de vaak uitgebreide verspreiding van de software.

Om dit probleem in het geheel op te lossen zal je dus een combinatie moeten maken tussen veilig gebruik van software (Kijkend naar de geschiedenis van software) en ander soort software.

#### 1.1.8 De eis "Alle uitvoer moet normaal binnen 10 seconden gegeven worden" is om één van de volgende redenen fout. Welke?

- a Dubbelzinnig
- b Niet concreet
- c Tegenstrijdig

De eis is niet concreet genoeg, doordat alle uitvoer heel algemeen is. Een verbetering op deze eis zou zijn: "De klant moet binnen 10 seconden een bevestiging van zijn bestelling op het scherm zien."

#### 1.1.9 Wat mankeert er aan de eis: "Het bestand moet een afsluitteken bevatten."?

De eis is onduidelijk, doordat het afsluitteken niet is vastgesteld in de eis.

#### 1.1.10 Welke maatregelen kunnen positief of negatief werken op:

- a De correctheid
- b De beschikbaarheid
- c De herstelbaarheid
- a Regelmatige validatie en verificatie tijdens alle stappen van het project.
- b Het systeem zo laten functioneren dat, mocht er een storing plaatsvinden in een bepaald gedeelte, dat de rest van het systeem nog naar behoren blijft functioneren.
- c Optimalisatie van de programmatuur.

## 2 Week 2

### 2.1 Opdrachten bij hoofdstuk 3

#### 2.1.1 De projectkosten worden begroot op 100000euro. De winst wordt gesteld op 15%. Het risico dat men met dit project denkt te lopen, is gebaseerd op de ervaring dat 20% van dit soort projecten mislukken. De BTW bedraagt 19%. Hoeveel is de aanbestedingsprijs?

$$(100 * 1.15 * 1.2) * 1.19 = 164.20euro$$

#### 2.1.2 Wat zijn de verschillen tussen kosten en investeringen?

Kosten zijn uitgaven die direct van de winst mogen worden afgetrokken. Inversteringen daarentegen moeten over meerdere jaren worden afgeschreven. Die jaarlijkse afschrijven mogen wel als kosten worden opgevoerd.

### 2.1.3 Noem 3 investeringscriteria

- Maximale gemiddelde boekhoudkundige rendabiliteit
- Minimale terugverdienperiode
- Concurrentievoordeel krijgen

2.1.4 Men kan een productiemiddel huren voor de prijs van 6100 e per maand. Indien men dit productiemiddel voor 120000 e aanschafft, moet men 100 e per maand onderhoud betalen. De restwaarde is nihil.

- Bepaal het omslagpunt.
- Bereken het omslagpunt indien de discontovoet 1% per maand is.
- Welke financiële overweging kan bij deze keuze een belangrijke rol spelen?
- Het omslagpunt:

$$\frac{120000}{(6100 - 100)} = 20$$

Het omslagpunt ligt dus bij 20 maanden

- Het omslagpunt bij een discontovoet van 1% per maand

$$120000 * 1.01^{-20} = 98345.34$$

$$\frac{98345.34}{6100 - 100} = 16.39$$

Het omslagpunt ligt dan bij 16.36 maanden.

- Welke overweging kan bij deze keuze een belangrijke rol spelen?  
De duurzaamheid of het gebruik van de investering. Hieruit bepaal je dan of het goedkoper is om te kopen of juist andersom.

## 3 Week 3

### 3.1 Opdrachten bij hoofdstuk 4

3.1.1 Noem minimaal 7 factoren die de individuele productiviteit beïnvloeden.

- 1 Kennisniveau
- 2 Sfeer
- 3 Taalbeperkingen
- 4 Afhankelijk van hulpmiddelen
- 5 Budget
- 6 Werkprocessen
- 7 Beloning

3.1.2 Wat is het bezwaar tegen de definitie van individuele productiviteit: “De omvang van de objectcode (gecompileerde programmatuur) in bytes per tijdseenheid”?

Iemand die minder efficiënt programmeert levert volgens deze stelling beter werk af. Ook wordt er hiermee niet gekeken naar de complexiteit van de code.

3.1.3 Verklaar waarom het coderen in de uitwerkingsfase meestal minder dan 10% van de totale kosten van de levenscyclus bedraagt.

Door het gebruik van hulpmiddelen krimpt het coderingsaandeel.

3.1.4 Verklaar hoe uit de COCOMO methoden blijkt dat de projecttijd niet afhankelijk is van het aantal programmeurs.

De geschatte projecttijd  $T$  wordt berekend door het aantal mensmaanden tot de macht  $c$  (Compactheid), vermenigvuldigt met 2,5. Op basis hiervan wordt het minimum aantal benodigde mensen geschat. Het model rekent niet de projecttijd uit op basis van het aantal teamleden.

3.1.5 Een administratief systeem van 9 netto functiepunten werd met een inspanning van 3 mensmaanden ontwikkeld. Is dit kenmerkend voor een administratief automatiseringsproject?

Volgens figuur 4.6 (Tabel 1) in de reader is de gemiddelde productiviteit voor administratieve systemen 15,2. De parameters van vraag 5 geven een productiviteit van 3.

Tabel 1: Figuur 4.6

Toepassingsgebied	Productiviteit P [nfp/mensmaand]		
	gemiddeld $\mu$	standaarddeviatie $\sigma$	variatiecoëfficiënt $\rho$
systemen met microcode	1,5	2,7	1,80
ingebodde realtime systemen	6,8	3,1	0,46
vliegtuigsystemen	6,4	3,3	0,52
commando en controle systemen	9,9	4,1	0,41
procesbesturings systemen	10,3	4,3	0,42
besturingsprogrammas en utilities	11,3	3,9	0,35
wetenschappelijke systemen	12,5	4,0	0,32
netwerksystemen	10,3	3,1	0,30
administratieve systemen	15,2	3,8	0,24

3.1.6 Een computerprogramma heeft bij functiepuntanalyse voor de systeemkarakteristieken een totale correctiefactor  $\bullet \sum_{i=1}^{14} c_i = 50$ . Uit de specificatie blijkt dat er 3 verschillende invoer-, 7 uitvoer- en 5 vraagtypen nodig zijn. Daarnaast zijn er 2 externe gegevensverzamelingen en 2 interne gegevensverzamelingen nodig. Alle geruiktsfuncties hebben een gemiddelde complexiteit. Men zal het computerprogramma coderen in de taal JAVa ( $C_t = 53$ ). Er wordt geen gebruik gemaakt van hergebruik. Maak een schatting voor de broncode.

$$f = 3 * 4 + 4 * 5 + 5 * 4 + 2 * 10 + 2 * 7 = 12 + 20 + 20 + 20 + 14 = 86$$

$$NFP = (0,65 + 0,01 * 50) * 86 = 98,9$$

$$S = 98,9 * 53 = 5241,7$$

Het aantal regels broncode is dus ongeveer 5241

3.1.7 Verklaar waarom juist computertalen met een lage  $c_t$  waarde (zie paragraaf 4.1.1) geschikt zijn voor prototyping.

Dit komt doordat de hoeveelheid code kleiner is en hierdoor dus sneller is aan te passen

### 3.1.8 Gegeven een studentinformatiesysteem bestaande uit de volgende onderdelen:

soort	omschrijving	complexiteit
module	inschrijving student	gemakkelijk
module	uitschrijving student	gemakkelijk
module	rekening collegegeld sturen	gemakkelijk
module	machtiging betaling collegegeld verwerken	makkelijk
module	betaling collegegeld verwerken	gemiddeld
module	aanmaningen sturen	gemakkelijk
module	tentamencijfers verwerken	moeilijk
module	studieresultaten naar Informatie Beheer Groep	gemiddeld
scherm	inschrijven van een student	gemiddeld
scherm	uitschrijven van een student	gemiddeld
scherm	betaling student invoeren	gemiddeld
scherm	machtiging betaling invoeren	gemiddeld
scherm	tentamencijfers invoeren	moeilijk
scherm	vakkentabel invoeren	gemiddeld
scherm	vakkentabel wijzigen	moeilijk
formulier	studentgegevens	gemiddeld
formulier	jaarlijkse voortgangsrapportage	moeilijk
formulier	rapportage van tentamenresultaten	moeilijk
formulier	aanmaningen collegegeld	gemiddeld

- Maak met een objectpuntanalyse (OPA) een schatting voor de inspanning E als het hergebruik rond de 30% ligt en het ontwikkelteam een lage productiviteit heeft. Maak een schatting met COCOMO-81 van de projecttijd T, indien het project een 'organische' karakter heeft.
- Maak een schatting van de projecttijd T met COCOMO-II, indien er geen sprake is van tijdsdruk en het project de volgende karakteristieken heeft:

Karakteristieken	$b_i$
ontwerpervaring	0,05
ontwerpvrijheid	0,02
ontwerprisico	0,3
ontwerpteam	0,03
organisatie	0,01

- OPA

$$NOP = 50 * (1 - 0.3) = 35$$

$$E = \frac{35}{7} = 5$$

- COCOMO-81

$$T = 2.5 * 5.2^{0.38} = 4.78$$

- COCOMO-II

$$T = (2.5 * 5.2^{0.33+0.2*0.14}) = 4.51$$



## 4 Week 4

### 4.1 Opdrachten bij hoofdstuk 5

4.1.1 Gegeven een netwerk met 7 activiteiten:

activiteit	omschrijving	afhankelijk van	$t_b$	$t_m$	$t_w$
A	Definitie		1	2	3
B	Hardware specificatie	A	3	4	5
C	Software specificatie	A	1	2	9
D	Hardware ontwikkeling	B	1	2	3
E	Software ontwikkeling	B C	4	5	12
F	Documenteren	D	3	4	5
G	Systeemintegratie	D E	3	3	3

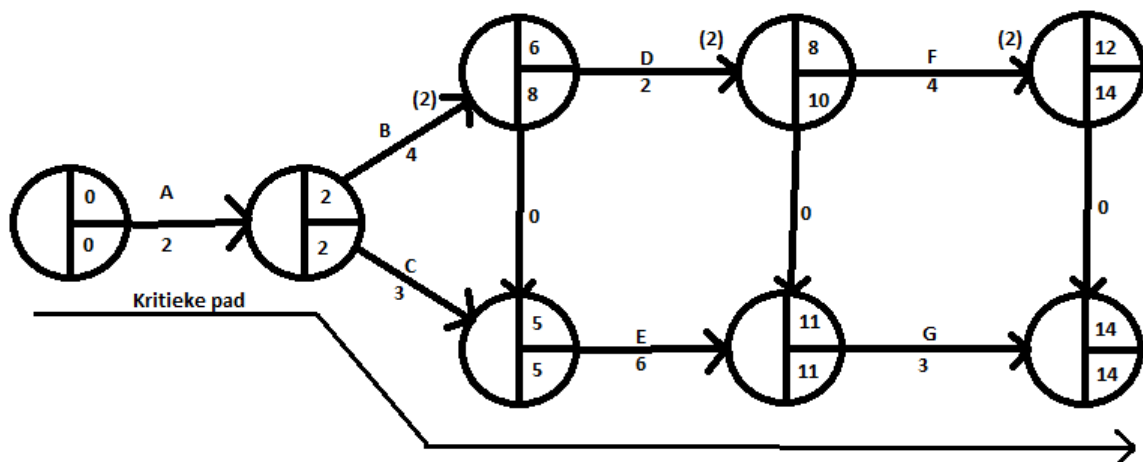
- Bepaal met PERT het mijlpalenplan en het kritieke pad
  - Bepaal het Gantt-scheme. Welke conclusies kan men daaruit trekken?
  - Bepaal het 95%-betrouwbaarheidsinterval voor de werkelijke projecttijd.
- Zie figuur 1.
  - Zie tabel 2.

$$\sigma^2 = 2 + 2 + 8 + 2 = 2 + 8 + 2 + 0 = 24$$

$$14 - (1,96 * \sqrt{24}) \leq T \leq 14 + (1,96 * \sqrt{24})$$

$$4,40 \leq T \leq 26,60$$

Figuur 1: Antwoord bij Hoofdstuk 5 Vraag 1.a



Tabel 2: Antwoord bij Hoofdstuk 5 vraag 1.b

	0..1	1..2	2..3	3..4	4..5	5..6	6..7	7..8	8..9	9..10	10..11	11..12	12..13	13..14
A 2	—	—												
B 4			—	—	—	—	---	---						
C 3			—	—	—									
D 2							---	---						
E 6						—	—	—	—	—	—			
F 4									---	---	—	—	---	---
G 3												—	—	—

4.1.2 bepaal van de volgend activiteiten met PERT het mijlpalenplan en een schatting van het kritieke pad  $T_k$

activiteit	$t$	afhankelijk van
A	2	K
B	3	H
C	4	
D	2	F G
E	1	D I J
F	3	B
G	3	C K
H	3	
I	2	A F
J	1	F
K	2	H

4.1.3 Van een collectief team is gegeven dat de verliesfactor per communicatiekanaal 5% is, bereken de maximale groepsgrootte

$$N = \frac{(1+i)}{(2*i)} = \frac{1+0,05}{2*0,05} = \frac{1,05}{0,10} = 10,5$$

4.1.4 Toon aan dat de volgende formule geldt voor het hiërarchieke team met  $N$  teamleden en maximaal 6 ondergeschikten per echelon:

$$\lceil \log(5N+1) \rceil \leq echelons \leq N$$

De 6 in de log geeft aan hoeveel ondergeschikten er in een echelon zijn. Wanneer je de 6 verandert naar een hoger of lager getal zal de waarde van  $N$  veranderen naar een nieuw maximaal aantal wat werkbaar is binnen een team. Tevens verandert hierin het aantal echelons ook negatief, wat betekent dat de communicatie en informatie overdracht tussen de verschillende echelons wordt vergroot. Alle informatie moet tenslotte eerste naar de bovenste echelon welke hier dan weer de informatie teruggeeft naar de ondergeschikte welke de benodigde informatie heeft bij één van zijn ondergeschikte. Wanneer dit niet via de bovenste echelon zou gaan zou de bovenste echelon niet de benodigde informatie hebben welke hij nodig heeft om het project goed te kunnen leiden. Het gebruik van de hiërarchieke structuur heeft grote voordelen bij grote projecten met veel verschillende bedrijven/onderdelen, doordat er een globaal overzicht is over hoever het project gevorderd is. Het nadeel blijft ook hier dat de informatie voorziening lastig kan zijn bij een groot aantal echelons.

## 5 Week 5

### 5.1 Opdrachten bij hoofdstuk 7

5.1.1 Van een objectgeoriënteerd computerprogramma in Java, zijn van alle classes de metrieken NOM, CBO, RFC, WMC, DIT en NOC gemeten. Een tiental classes had een verhoogd risico.

a Maak een tabel met riskante waarden van de waarden van de metrieken.

Metrieken	Riskante waarde	reader bladzijde	reader paragraaf
NOM	Java gemiddeld: $\approx 8$ C++ gemiddeld: $\approx 25$ Kritiek: $> 40$	84	7.2.2
CBO	5	86	7.2.5
RFC	$\geq 50$	87	7.2.7
RFC/NOM	Java: $\geq 10$ C++: $\geq 5$	88	7.2.7
WMC	aanbevolen: $\geq 25$ kritiek: $> 75$	83	7.2.1
DIT	5	84	7.2.3
NOC	Geen aanbevolen waarde, hoge NOC geeft slechte metriek	85	7.2.4

b Geef in de volgende tabel bij elke class aan welke metriek een kritieke waarde heeft.

class	NOM	CBO	RFC	RFC/NOM	WMC	DIT	NOC
1	54	8	536	9,9	175	1	0
2	7	6	168	24,0	71	4	0
3	33	4	240	7,2	105	2	0
4	54	8	381	6,7	117	2	2
5	62	6	378	6,1	163	2	0
6	63	7	235	3,7	156	2	0
7	81	10	285	3,5	161	2	0
8	42	5	127	3,0	69	3	0
9	20	17	325	16,2	139	4	4
10	46	5	186	4,0	238	1	3

Bij NOC is er geen aanbevolen kritieke waarde, maar er is wel hoe hoger deze waarde is hoe slechter de metriek is. Wij hebben bij een NOC van  $\geq 3$  aangenomen dat de NOC kritiek is. Dit baseren wij op dat de NOC is gebaseerd op het aantal kinderen een class heeft, en hoe meer kinderen, hoe groter de kans op fout. Bij  $\geq 3$  is de kans op fouten volgens onze mening zeer goed aanwezig.

Wanneer een waarde kritiek is voor een bepaald metriek, staat er in onderstaande tabel *kritiek*. Wanneer er niets staat is de waarde niet kritiek. Er is uitgegaan dat de classes zijn geschreven in de taal JAVA, zoals vermeld in de opdracht. Echter indien de waarde uitmaakt in welke taal (JAVA of C++) de class is geschreven voor de kritieke waarde, staat de waarde voor C++ tussen haakjes.

De waarde voor RFC is in alle gevallen kritiek. De waarde ligt flink boven de aangegeven kritieke waarde, welke tevens ook zeer hoog boven de kritieke waarde ligt. De kritieke waarde voor RFC is  $\geq 50$ , terwijl de laagste waarde voor RFC van de classes 127 is.

De waarde voor DIT ligt in alle gevallen onder de kritieke waarde van 5.

class	NOM	CBO	RFC	RFC/NOM	WMC	DIT	NOC
1	kritiek	kritiek	kritiek	(kritiek)	kritiek		
2		kritiek	kritiek	kritiek(kritiek)			
3			kritiek	(kritiek)	kritiek		
4	kritiek	kritiek	kritiek	(kritiek)	kritiek		
5	kritiek	kritiek	kritiek	(kritiek)	kritiek		
6	kritiek	kritiek	kritiek	(kritiek)	kritiek		
7	kritiek	kritiek	kritiek		kritiek		
8	kritiek	kritiek	kritiek				
9		kritiek	kritiek	kritiek(kritiek)	kritiek		kritiek
10	kritiek	kritiek	kritiek		kritiek		kritiek

5.1.2 Bepaald WMC, DIT, NOC, CBO, RFC en LCOM van de volgende pseudo objecten broncode:

Class Variables Methods	trade trade.id, counterparty, trade_value evaluate_counterpart() get_trade_id(trade_id) position_update() {position_manager::report_trade()}
Class Variables Methods	bond_trade bond_detials get_bond_info()
Class Variables Methods	fx_trade forex_detials calculate_exchange_rates()
Class Variables Methods	equity_trade company, stock_market, PE_ratio, earnings, week_hi_lo Estimate_beta() get_stock_quotes() {quotron::quotes()}
Class Variables Methods	municipal_bond_trade state_or_federal, over_the_counter calculate_coupon_rate() {Tbil_server::rate()}
Class Variables Methods	corporate_bond_trade adr, sp_rating calc_rating(sp_rating) {if adr then fx_trade::calculate_exchange_rates()}
Class Variables Methods	international_equity exchange_rate, quotation perform_anaylysis_roa() {fx_trade::calculate_exchange_rates()} get_quotron(quotation)
Class Variables Methods	domestic_equity variables: attribute1

In de tabel hieronder staan alle waarde voor de verschillende metrieken. Onder de tabel staat de uitleg erbij met de berekening van de verschillende metrieken. Tevens staat in de tabel hieronder of de waarde kritiek is voor het type metriek.

Metriek	Waarde	Kritek
WMC	11	Nee
DIT	Zie hieronder	
NOC	Zie hieronder	
CBO	4	Nee, maar zie hieronder
RFC	16	Nee, maar zie hieronder
LCOM	Zie hieronder	

## WMC

WMC houdt in dat er voor iedere methode in een class 1 wordt geteld. Voor iedere routine in die methode word daarbij nog 1 opgeteld. Er is in dit geval maar bij één class in een methode een routine aanwezig.

### **Class trade**

De WMC van de class trade is als volgt:

- 3 methodes: 3
- Geen routines

De WMC van de class trade is 3.

### **Class bondtrade**

De WMC van de class bondtrade is als volgt:

- 1 methode: 1
- Geen routines

De WMC van de class bondtrade is 1.

### **Class fx\_trade**

De WMC van de class fx\_trade is als volgt:

- 1 methode: 1
- Geen routines

De WMC van de class fx\_trade is 1.

### **Class equity\_trade**

De WMC van de class equity\_trade is als volgt:

- 2 methodes: 2
- Geen routines

De WMC van de class equity\_trade is: 2.

### **Class municipal\_bond\_trade**

De WMC van de class municipal\_bond\_trade is als volgt:

- 1 methode: 1
- Geen routines

De WMC van de class municipal\_bond\_trade is: 1.

### **Class corporate\_bond\_trade**

De WMC van de class corporate\_bond\_trade is als volgt:

- 1 methode: 1
- 1 routine: 1

De WMC van de class corporate\_bond\_trade is:  $1 + 1 = 2$ .

### **Class international\_equity**

De WMC van de class international\_equity is als volgt:

- 1 methode: 1
- geen routines

De WMC van de class international\_equity is: 1

### **Class domestic\_equity**

De WMC van de class domestic\_equity is als volgt:

- Geen methode
- geen routines

De WMC van de class `domestic_equity` is: 0

#### **Totale WMC**

De totale WMC is als volgt:

$$3 + 1 + 1 + 2 + 1 + 2 + 1 = 11$$

#### **DIT**

DIT houdt in dat er wordt gekeken naar de overerving van classes. Hierbij wordt het aantal ouders geteld wat een bepaalde class is. De DIT hoort niet hoger te zijn als 5 voor een bepaalde class. Bij de DIT wordt puur per class gekeken, en niet zoals bij de WMC naar het totaal. In het geval van de opgegeven pseude code is niet duidelijk uit te halen welke classes een ouder hebben, en wat de ouder dan ook is. Hierdoor is het niet mogelijk de DIT van de classes te berekenen. Mogelijk dat alle classes in dit geval super classes (Ofwel zonder ouders zijn). Dit betekend echter wel dat de classes slecht zijn opgebouwd. Maar ook hiervoor is geen direct bewijs uit de pseude classes te halen.

#### **NOC**

NOC geeft een indicatie wat een bepaalde class voor uitoefining heeft op zijn kinderen. Hierbij geeft een hoge NOC aan dat de classes slechter te testen zijn. Er is geen vaste waarde voor de NOC welke slecht is, maar een hogere NOC kan problemen opleveren. Net als bij de DIT is de NOC afhankelijk van de kinderen/ouders. Doordat dit niet is aangegeven in de pseudocode is het niet te berekenen wat de NOC is. Mogelijk zijn er geen kinderen, en is de NOC 0.

#### **CBO**

Bij CBO wordt gekeken naar afhankelijkheid van andere classes welke niet door overerving gekoppeld zijn. Doordat in dit geval geen relates zijn opgegeven in de pseude code gaan we ervan uit dat alle methode calls welke uitgevoerd worden niet door overerving zijn bedoeld. De volgende classes hebben geen methode calls buiten hun eigen class, en de CBO is dus 0:

- `bond_trade`
- `fx_trade`
- `dometic_equity`

De volgende classes hebben allemaal één methode call buiten hun eigen class:

- `equity_trade`
- `municipal_bond_trade`
- `corporate_bond_trade`
- `international_equity`

De CBO van de classes is totaal bij elkaar dus 4. De CBO hoort in principe  $< 5$  te zijn, algemeen aangenomen. Dit is het geval in de bovenstaande classes. Bij een hogere CBO wordt de koppeling groot, en is de modulariteit van het ontwerp in het geding. Dit kan betekenen dat de classes lastig te zijn hergebruiken, terwijl juist de bedoeling van het gebruik van classes is dat deze goed te zijn hergebruiken. In bovenstaand voorbeeld kom ik in dit geval tot een andere conclusie. Doordat er, zover te zien in de pseudocode, geen overerving is, zijn alle classes afhankelijk van elkaar, dit zorgt er in dit specifieke geval dus voor dat één losse klas eigenlijk helemaal niet makkelijk los te gebruiken is. Dit komt doordat de CBO gemaakt is voor grotere stukken code, terwijl het voorbeeld hier zeer minimaal is. Volgens de standaard is de code dus eigenlijk niet kritiek, maar ik durf te zeggen dat deze dat wel is.

#### **RFC**

De RFC telt het aantal methode in een class, plus het aantal externe methode welke aangeroepen worden. Het resultaat van de metriek is de intensiviteit van de communicatie met andere classes. Het is aan te raden om de RFC op  $\leq 50$  te houden. Hierboven wordt de waarde van de complexiteit te hoog waardoor het lastig wordt om de class te wijzigen en de class te snappen.

De volgende classes hebben geen externe methode welke aanroepen. Hierbij is de RFC van die class gelijk aan het aantal methode:

- `bond_trade`: 1 methode
- `fx_trade`: 1 methode
- `domestic_equity`: 0 methodes

De RFC van deze 3 classes is dus bij elkaar 2. De volgende classes hebben naast normale methode ook methode welke extern aanroepen worden. Hierbij is de RFC van die class gelijk aan het aantal methoden plus het aantal extern aanroepen methoden. Het aantal externe methodes welke aanroepen worden is in alle gevallen één.

- `trade`: 3 methodes
- `equity_trade`: 2 methodes
- `municipal_bond_trade`: 1 methode
- `corporate_bond_trade`: 1 methode
- `international_equity`: 2 methodes

In totaal is de RFC voor deze 5 classes:  $3 + 2 + 1 + 1 + 2 + 5 = 14$ . De 5 in de berekening komt van de extern aanroepen methode. In totaal is de RFC van alle classes totaal  $2 + 14 = 16$ . De kritieke waarde voor RFC van classes is  $\leq 50$ , en volgens de eigenlijke standaard is de RFC niet kritiek. Maar net als bij de CBO durf ik te stellen dat in dit geval de pseudo code niet duidelijk genoeg is, waardoor de eigenlijke complexiteit van de classes toch te hoog is, terwijl de RFC toch binnen de aangegeven waarde valt. Wanneer er duidelijker in de pseudocode wordt aangegeven wat de exacte classes zijn, en wanneer deze ook duidelijke referenties aangeven kan er een betere conclusie worden getrokken uit de waarde.

### LCOM

De LCOM geeft aan in hoeveel een class met elkaar in verband staan. Hierbij wordt gekeken naar de gemeenschappelijk object variable. Hoe lager de LCOM, hoe beter de class samenwerkt. Wanneer de LCOM te hoog is, is de samenhang niet goed en moet de class eigenlijk gesplitst worden in meerdere losse classes. In het meest gunstige geval is de  $LCOM \leq 1$ , wat een goede samenhang betekent.

Doordat de pseudocode niet aangeeft welke object variable gebruikt worden in een bepaalde methode is het niet te doen om de LCOM te berekenen. Om de LCOM te kunnen berekenen zal er in de pseudocode moeten worden weergegeven welke object variable er in die methode gebruikt wordt.

### Conclusie

Zoals hierboven al geschreven kunnen de DIT, NOC en LCOM niet berekend worden doordat de vereiste gegevens missen in de pseudo code. Wanneer de vereiste gegevens wel gegeven waren konden deze drie metrieken vrij eenvoudig berekend worden. Echter, om tot een goed resultaat te komen, zoals bij een normaal project echt gebruikt wordt zal de code welke gebruikt wordt om deze opdracht te maken flink uitgebreid moeten worden. De huidige resultaten uit de diverse metrieken komen niet goed overeen met de eigenlijke complexiteit van de gegeven code. Dit komt doordat de code welke aanwezig is zo klein is dat je weinig complexiteit ziet in de metrieken. Maar wanneer je gaat kijken naar de diverse methode en classes hoe ze in elkaar zitten zie je dat alle classes eigenlijk afhankelijk zijn van elkaar. Om deze redenen zijn de classes dus niet correct opgebouwd en kan de opgegeven classes niet hergebruiken, terwijl dit uiteindelijk wel het originele doel is van het gebruik van classes. Om dit toch te kunnen bewerkstelligen zouden alle aanwezige classes welke er zijn herschreven moeten worden, zodat ze data welke universeel is accepteren. Tevens zouden de extern aanroepen methode verkleint moeten worden met het huidige aantal. Wanneer dit gebeurd is zullen de classes uit de pseudocode niet alleen beter uit de verschillende metrieken komen, maar ook beter herbruikbaar zijn voor de verschillende opdrachten.