Software Engineering

Assignment 5: Software Metrics

3 Ba INF 2018-2019

Benjamin Vandersmissen
Franciscus Fekkes

December 16, 2018

1 Complexiteit

Om de complexiteit van het project te bepalen zijn de externe in- en outputs bepaald, de inquiries, externe interfaxes en logical files. Daarna kunnen deze onderverdeeld worden in hoe moeilijk het is om ze te implementeren: simpel (S), gemiddeld (A) en complex (C). Ieder van deze categorieën krijgt een gewicht om zo een numerieke waarde te kunnen bekomen voor de complexiteit van het hele programma.
De evaluatie van de functionaliteiten en datastructuren is te vinden in tabel 1. De datastructuren zijn opgedeeld in Interface Logical Files en External Interface Files. De eerste soort wordt alleen door de user (clients en administrators) aangepast en de tweede wordt door externe bronnen aangepast. De functionele aspecten worden opgedeeld in drie delen: External Inputs, External Outputs en External Inquiries. External Inputs laten de users toe om input te geven aan het systeem. External Outputs genereren data/informatie. External Inquiries laten de users toe om selecte data op te vragen.
Nadat alle functionalieten en datastructuren zijn opgedeeld, kan de som van de unadjusted function points berekend worden. Deze zal dan vermenigvuldigd worden met de complexity factor die berekend is in tabel 3.
Met de formule $\#LOC/AFP = 46$ voor java kunnen we dan een gerichte schatting maken van het aantal lijnen code dat het programma in totaal zal hebben. Dit zal waarschijnlijk rond de 8400 zijn. Dit zal waarschijnlijk aan de hoge kant liggen.

2 Analyse Source Code

Met de tool PMD is de source code nog geanalyseerd op grote en kleine problemen. Niet alle fouten zullen tot in detail besproken worden maar alles wordt

eerder oppervlakkig en in het algemeen bekeken.
Vervang de oproepen waarbij $size() == 0$ of $size() > of < 1$ wordt bekeken met een functie $isEmpty()$.
Zorg voor duidelijkere namen voor variabelen zoals voor de variabele m_db en m_id. Deze namen geven soms niet duidelijk weer waarvoor ze staan.
Vermijdt zeer lange namen zoals $prev\_iter\_itemsets$.
Constructors zouden geen exceptions moeten throwen zoals in $TransactionDB.java$.
Vermijdt ook nested if statements zoals in $Cart.java\ remove\_item$.
Voor performance redenen zouden new objecten buiten lussen aangemaakt moeten worden ($Eclat.java$ & $TransactionDB.java$).
Er mist nogsteeds een hele hoop documentatie. Er missen header comments, method en constuctor comments etc...

3 Refactoring

Er zijn 4 manieren om aan refactoring te doen: Extract Method, Move Behaviour Close to the Data, Eliminate Navigation Code and Transform Self Type.

3.1 Extract Method

Deze refactoring gaat duplicate code vervangen door een functie die meerdere malen gebruikt kan worden. De werkwijze bestaat uit 3 delen:

- Zoek naar duplicate code
- Baken de verschillen af in duplicate code door extra variabelen te introduceren
- Zet de duplicate code in een functie en pas die functie toe met de juiste variabelen

3.2 Move Behaviour Close to the Data

Als een methode alleen maar data gebruikt uit een specifieke klasse, maar deze methode staat buiten de klasse (bijvoorbeeld in een andere klasse, of als onafhankelijke functie), dan wordt het encapsulatie principe niet gevolgd. Dit principe zegt dat methodes op specifieke data gebundeld moeten worden met die data. De refactoring houdt dan in dat we een methode verplaatsten naar de klasse waar de data inzit waar die methode op inwerkt.

3.3 Eliminate Navigation Code

Methodes buiten een klasse moeten vermijden de interne structuur van een klasse te kennen. Dit wil zeggen dat ze niet de interne variabelen rechtstreeks mogen aanroepen, maar altijd zoveel mogelijk via methoden op de klasse moeten werken. Dit is om *change dependencies* te vermijden. Een *change dependency*

Data structure:	Complexiteit:	Type:
Cart	A	ILF
Catalog	C	ILF
Category	S	ILF
Item	C	ILF
Order	A	ILF
Orders	A	EIF
Client	A	ILF
Admin	A	ILF

Functionaliteit:	Complexiteit:	Type:
user sort	A	EQ
user add items cart	A	EI
user remove items cart	A	EI
user chanhe desired quantity cart	A	EI
user order items cart	A	EO
user enter standard info	A	EI
user select payment	S	EI
user cancel order not ready delivery	C	EI
user overview personal info	S	EO
user overview mailing preferences	S	EO
user overview own open/delivered orders	S	EQ
user overview popular items	S	EO
user sort items name/pop/price	A	EQ
user/admin display catalog	S	EO
user/admin order by category	A	EQ
admin add category	A	EI
admin change placed order	A	EI
admin overview all open/deliverd orders	A	EQ
admin check statistics	C	EQ
admin analysis previous orders	C	EQ

Table 1: Evaluatie complexiteit en type van alle datastructuren en functionaliteiten. Complexity: S: Simple, A:Average, C:Complex. Types:ILF: Interface Logical Files, ELF: External Intergace Files, EI: External Inputs, EO: External Outputs, EI: External Inquiries.

	Weighting Factor			
Item	Simple	Average	Complex	sum:
External Inputs	1 * 3 = 3	6 * 4 =24	1 * 6 = 6	33
External Outputs	4 * 4 =16	1 * 5 = 5	0 * 7 = 0	21
Inquiries	1 * 3 = 3	4 * 4 =16	2 * 6 = 12	31
External Interfaces	0 * 5 = 5	1 * 7 = 7	0 *10 = 50	7
Logical Files	1 * 7 = 7	4 *10 =40	2 *15 = 30	77
Unadjusted Function Points				169
Adjusted Function Points	x Complexity Factor (1.075)			181.675

Table 2: Tabel met weightig factors waarmee de unadjusted function points worden berekend. De complexity factor is berekend in tabel 3.

Complexity factor	**Rating(0..5)**	**Weight**	**Total**
Distributed System	5	*2 =	10
Performance objectives	4	*1 =	4
End-use efficiency	4	*1 =	4
Complex internal processing	3	*1 =	3
Code must be reusable	2	*1 =	2
Easy to install	2	*0.5=	1
Easy to use	1	*0.5=	.5
Portable	2	*2 =	4
Easy to change	3	*1 =	3
Concurrent	3	*1 =	3
Special security	4	*1 =	4
Direct access for 3rd parties	3	*1 =	3
Special user training	2	*1 =	2
Total Complexity			=sum(Total)=42.5
Complexity factor			=0.65+Total Complexity*0.01=1.075

Table 3: test

betekent dat als iets in de klasse aangepast wordt, dat dit ook op andere plaatsen moet aangepast worden, omdat de interne structuur die gerefereerd werd, aangepast werd. De code die *change dependencies* introduceert, wordt ook wel *navigation code* genoemd. Het refactoring proces bestaat uit 2 delen:

- verplaats de navigation code naar een aparte method
- voeg de method toe in de relevante klasse

Merk op dat het eerste deel van deze werkwijze sterk lijkt op *Extract Method* en het tweede deel is hetzelfde als *Move Behaviour close to the Data*

3.4 Transform Self Type Checks

Self type checks wilt zeggen dat er in een klasse een variabele aanwezig is, vaak type of gelijkaardig genoemd, die verschillende waardes kan aannemen en afhankelijk van die waardes verschillende functionaliteit kan uitvoeren. Dit kan logisch beschouwd worden als subclasses van een gedeelde basis klasse. Het doel van deze refactoring is het overbodig maken van de type variabele en de code herverdelen in subclasses. Hiervoor kunnen we de volgende stappen toepassen:

- Maak subclasses aan voor de oorsponkelijke klasse, 1 voor elke mogelijke waarde van type
- Gebruik de subclasses daar waar nodig is in de code ipv de oorspronkelijke klasse
- Bekijk voor elke functie die de type variabele gebruikt, welke code naar welke subclass verplaatst mag worden. Dit doen we door de methode virtual te maken in de basis klasse en die te overschrijven in de subclasses.
- Verwijder de nu overbodige type checks uit de code in de subclasses.

3.5 Examples of refactoring methods in given code

We kunnen **Transform Self Type Checks** gebruiken in de klasse *Payment.java*. Deze klasse heeft een variabele *type* en op basis van die variabele wordt er verschillende functionaliteit geimplementeerd in *PaymentProcessor*.

Een andere refactoring methode die we kunnen toepassen, is op *PaymentProcessor*. Deze klasse heeft functionaliteit die alleen maar inwerkt op de klasse *Payment*. We kunnen deze code dus verplaatsen naar de klasse *Payment* (en zijn subclasses) door **Eliminate Navigation Code** toe te passen.

Uiteindelijk kunnen we de hele class *PaymentProcessor* wegwerken met de refactoring, door ook **Extract Method** en **Move Behaviour Close to the Data** toe te passen.