# Verslag opdracht 1 Software Evolution

### Samenwerking

Marco en Ivo hebben regelmatig via mail, WhatsApp en telefonisch overlegd. Tijdens deze contactmomenten is o.a. besproken op welke wijze de metrieken precies geïmplementeerd moeten worden. Marco heeft de metrieken “Complexity” en “Unit size” uitgewerkt en Ivo heeft “Duplication”, “Volume” en “Unit testing” voor zijn rekening genomen. Marco en Ivo hebben samen aan het verslag gewerkt

### Inhoud

Dit verslag presenteert de resultaten en conclusie van de analyse van de bepaling van de software onderhoudbaarmetrieken van de systemen SmallSQL en HyperSQL Bijlage A bevat aanvullende informatie voor enkele metrieken. Bijlage B bevat de output van de processen voor beide systemen.

### Aannames

De volgende aannames zijn gemaakt:

1. Voor het bereken van de metriek “Complexity per unit” en “size of each unit” wordt een methode in een Java klasse als unit beschouwd. Immers een methode is het kleinst mogelijke uitvoerbare onderdeel.
2. T.b.v. de bepaling van de metrieken voor HyperSQL, is alleen de directory “src” gebruikt. Er bevinden zich namelijk ook Java bestanden in de directory “doc”. Deze zijn niet meegenomen in de bepaling van de metrieken.

### Duplication

In [[1]](#footnote-1) wordt er een oplossingsmethode voorgesteld om de metriek “Duplication” van een systeem te bereken waarbij blokken van 6 regels met elkaar worden vergeleken. Men moet zich realiseren dat indien men gebruik maakt van deze methode, er rekening mee moet worden gehouden dat alleen “concrete duplications” gevonden zullen worden. Stukken code die logisch gezien hetzelfde zijn, maar verschillende namen van bijvoorbeeld attributen gebruiken, zullen op deze manier niet als duplications worden aangemerkt. De implementatie verwijdert lege regels, blokcommentaar, regelcommentaar, import statements en voert ook een compressie uit door alle whitespace te verwijderen voordat met de duplicatie detectie gestart wordt. De resultaten voor de systemen HyperSQL en SmallSQL zijn de volgende:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **System** | **Metriek** | **Rank** | **System** | **Metriek** | **Rank** |
| SmallSQL | 10.1% | - | HyperSQL | 14.7% | - |

Beide systemen hebben een rank van “-“ voor duplication. Zoals ook uit figuur 2 van Bijlage A blijkt, vlakt het histogram van HyperSQL niet snel af: Veel methoden hebben een duplication percentage in het bereik 10-25. Voor beide systemen geldt dat er meerdere bronbestanden zijn die zelfs een duplication hebben van 70 á 80%.

### Volume

De metriek “volume” wordt bepaald door het aantal regels te tellen in de broncode. Behalve de verwijdering van import statements, worden dezelfde preprocessing methode toegepast als bij de bepaling van de metriek “duplication”. De resultaten::

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **System** | **Metriek** | **Rank** | **System** | **Metriek** | **Rank** |
| SmallSQL | 24788 | ++ | HyperSQL | 163205 | + |

### Complexity

De metriek “complexity” wordt bepaald door het tellen van het aantal predicaten (b.v. if, while, and) en het aantal line of code (loc) per unit. Aan de hand van de complexity wordt aan de unit een bepaalde risk (b.v. moderate) toegekend[[2]](#footnote-2). Het aantal loc wordt per categorie opgeteld. Vervolgens kan per categorie het percentage t.o.v. het totaal aantal loc van de units worden uitgerekend. De laatste stap is het bepalen van de rank door het afzetten van de percentages per categorie t.o.v. de rank. Voor rank + mag de code voor b.v. voor maximaal 5% uit high risk units bestaan. De resultaten voor de systemen HyperSQL en SmallSQL zijn de volgende:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **System** | **Metriek** | **Rank** | **System** | **Metriek** | **Rank** |
| SmallSQL | 5 | -- | HyperSQL | 5 | -- |

### Unit size

Voor het bepalen van de waarde voor de metriek “size of each unit” is gebruik gemaakt van tabel IIIa in het artikel “Benchmark based Aggregation of Metric to Ratings”[[3]](#footnote-3). De resultaten voor de systemen HyperSQL en SmallSQL zijn de volgende:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **System** | **Metriek** | **Rank** | **System** | **Metriek** | **Rank** |
| SmallSQL | 4 | - | HyperSQL | 5 | -- |

Als eerste wordt voor een unit de line of code (loc) bepaald. De waarde van de loc bepaald de categorie (e.g. low risk) waarin de unit valt. Dit wordt voor alle units gedaan. Door per categorie het aantal locs op te tellen, kan het percentage loc per categorie berekend worden. In de genoemde tabel kan vervolgens de unit size rating worden afgelezen.

### Unit testing

De implementatie van de metriek “Unit testing” is gedaan op basis van de beschrijving van het artikel “A Practical Model for Measuring Maintainability” [[4]](#footnote-4). Hier wordt vermeld dat de beste manier voor het bepalen van deze metriek, m.b.v. een dynamische analyse gedaan moeten worden. Om toch een poging te doen de metriek te bepalen, is er bij de implementatie gekeken naar het aantal assert statements in de code. Idealerwijs, moet het aantal assert statements groter dan of gelijk zijn aan het aantal paden (dus McCabe waarden) van alle methoden in het systeem. De tabel die genoemd staat onder optie 1 (Unit test coverage) is ook gebruikt in de implementatie om deze metriek op een statische manier te bepalen. De resultaten voor de systemen HyperSQL en SmallSQL zijn de volgende:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **System** | **Metriek** | **Rank** | **System** | **Metriek** | **Rank** |
| SmallSQL | 0 | -- | HyperSQL | 5 | -- |

Voor beide systemen worden geen assert statements gevonden. Dit is de reden voor de slechts mogelijk score. Een statische methode om deze metriek te bepalen zal nooit betrouwbare resultaten geven, aangezien er veel verschillende soorten methoden van testen zijn, en veel verschillende soorten frameworks. Het is dus beter om deze metriek niet te gebruiken in de interpretatie.

### Validatie

### De berekening van de complexiteit van een unit is getest door middel van een test Java applicatie. Deze applicatie bevat diverse methoden met verschillende flow control elementen ( b.v. if, while, case). Per methode is aangegeven wat de verwachte complexiteit is. Door de metrieken van de test applicatie te berekenen kan bepaald worden of de Rascal metriek berekening correct is. Op analoge wijze wordt de unit size getest. Iedere metriek is ontwikkeld in een aparte Rascal module en de main methode van elke module fungeert als een Unit test. Zoals is vermeld is er een test applicatie voorbereid en er wordt in de unit test gechecked of het verwachte en werkelijke resultaat gelijk is.

### Interpretation

De SIG mapping voor SmallSQL ziet als volgt uit:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Volume | Complexity per unit | Duplication | Unit size | Unit testing |  |
| ++ | -- | - | - | -- |
| Analysability | x |  | x | x | x | -\0 | |
| Changeablity |  | x | x |  |  | -- | |
| Stability |  |  |  |  | x | -- | |
| Testability |  | x |  | x | x | -- | |

### De applicatie smallSql scoort slecht op alle vier de subcategorieën, waarbij de analysaibity het “beste” scoort. De complexiteit alsmede het aantal regels van te veel methoden is te hoog. Ook zijn er te weinig unittesten. Hierdoor is het lastig om wijzigingen door te voeren. Het aantal unit testen zal uitgebreid moet worden en de complexiteit van een aantal methoden zal moeten worden verminderd. Ook zal een refactoringslag gedaan kunnen worden, zodat de unit size van te grote methode verlaagd wordt.

HyperSQL heeft de volgende SIG mapping:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Volume | Complexity per unit | Duplication | Unit size | Unit testing |  |
| + | -- | - | -- | -- |
| Analysability | x |  | x | x | x | - | |
| Changeablity |  | x | x |  |  | -/-- | |
| Stability |  |  |  |  | x | -- | |
| Testability |  | x |  | x | x | -- | |

### De applicatie HyperSQL scoort ook slecht op alle vier de subcategorieën. Er zijn veel bronbestanden die een duplication hebben van meer dan 70%. Ook zijn er erg veel methoden met een unit size die meer dan 500 regels hebben. Dit betekent dat de testbaarheid en de onderhoudbaarheid erg in het geding komen. M.a.w. deze applicatie herbergt zeer veel “technical debt” en een grote refactoringslag is nodig om alle vier de subcategorieën naar een goed niveau te brengen.

### Bijlage A: Histogrammen

Deze bijlage bevat aanvullende informatie betreffende de metrieken die berekend zijn t.b.v. de bepaling van de onderhoudbaarheid van de twee systemen SmallSQL en HyperSQL.

Figuur 1: Duplication SmallSQL

Figuur 2: Duplication HyperSQL

Figuur 3: Complexity SmallSQL

Figuur 4: Complexity HyperSQL

Figuur 5: Unit size SmallSQL

Figuur 6: Unit size SmallSQL

### Bijlage B: Output

Deze bijlage de output van het proces van de bepaling van de metrieken van de systemen SmallSQL en HyperSQL. Alleen de meeste relevante informatie is hier getoond, detaillistische debug informatie is niet weergegeven. Deze zijn te vinden in de bestanden Metrics\_smallsql.txt en Metrics\_hsqldb.txt.

SmallSQL:

Metrics for system: smallsql

<"Volume",24.788,"++">

<"Duplication",10.1860920700,"-">

<"Unit Testing",0,"--">

<"Cyclomatic complexity",5,"--">

<"Unit size",4,"-">

HyperSQL:

Metrics for system: hsqldb

<"Volume",163.205,"+">

<"Duplication",14.7028469300,"-">

<"Unit Testing",0,"--">

<"Cyclomatic complexity",5,"--">

<"Unit size",5,"--">

1. Artikel 2 “A practical Model for Measuring Maintainability”, pagina 27 [↑](#footnote-ref-1)
2. Artikel 2 “A practical Model for Measuring Maintainability”, onderdeel ‘C Complexity per unit’ [↑](#footnote-ref-2)
3. https://www.sig.eu/files/en/080\_Benchmark-based\_Aggregation\_of\_Metrics\_to\_Ratings.pdf [↑](#footnote-ref-3)
4. Artikel 2 “A practical Model for Measuring Maintainability”, pagina 27 [↑](#footnote-ref-4)