# Automatisering van het tellen van functiepunten

Mirela Mustacevic

10 oktober 2000

# Voorwoord

## Abstract

Software metrieken spelen een grote rol in software engineering omdat ze ontwikkelingskosten, ontwikkelingstijd, de grootte van softwareprojecten e.d. kunnen bepalen. Één van de meest betrouwbare en de meest gebruikte meettechnieken is de functiepuntenmetriek die de omvang van bedrijfsapplicaties bepaalt op basis van aan de gebruiker geleverde businessfuncties. Maar helaas is het tellen van exact aantal functiepunten in al bestaande systemen een onplezierig en vaak ook langdurig werk omdat het handmatig gedaan moet worden.

Daarom hebben we een onderzoek gedaan naar het automatiseringsprobleem van de FP methode. Ons doel was het vinden van een methode die automatisch het aantal functiepunten uit broncode kon afleiden. Daarvoor is in het onderzoek gebruik van COBOL, SQL en CICS programmeertalen gemaakt. Uit bestaande sources worden eerst de statements geëxtraheerd die naar businessfuncties verwijzen. Daarna wordt er gekeken welke functie (invoer - IT, uitvoer - OT, navraag - QT, bestand - FT of interface - EI) die statements in de betreffende applicatie vervullen. Verder moet nog complexiteit van de gevonden functies bepaald worden en op basis daarvan zou een getal aan ieder functioneel object toegekend worden. Ten slotte worden die getallen gesommeerd en vermenigvuldigd met de som van al vastgestelde weegfactoren.

De boven beschreven methode is in het onderzoek toegepast op een applicatie waarvoor het aantal functiepunten al bekend is. Het verkregen resultaat wordt dan met het al berekende aantal functiepunten vergeleken net zoals met de backfiring methode die men gebruikt om snel een globale schatting van het aantal functiepunten in een applicatie te maken.

De uikomsten van verrichte tellingen laten ons zien dat het automatiseringsprobleem van de functiepuntenmethode oplosbaar is en dat er zelfs een snelle manier van tellen bestaat die betere resultaten oplevert dan de backfiring methode.

## Met dank aan

Ik wil mijn afstudeerdocent Paul Klint en mijn stagebegeleider Chris Verhoef bedanken voor hun hulp en adviezen gedurende mijn afstudeeronderzoek en de tijd waarin ik met het schrijven van deze scriptie bezig was. Verder dank ik ABN AMRO voor het honoreren van mijn onderzoek met de broncode die ik voor het meten nodig had. En ten slotte maar niet als het minst belangrijk wil ik de mensen van de Software Improvement Group bedanken voor het geven van een leuke werkplek en alle uitleg en hulp rondom DocGen.

# Inhoudsopgave

# Hoofdstuk 1

# Inleiding

Jarenlang was het meten van software een groot raadsel voor software engineers. Tegenwoordig wil men heel graag weten wat de grootte, kwaliteit, nodige ontwikkelingstijd van een software applicatie is. Als deze factoren bekend zijn, kan men verder bepalen hoeveel mensen aan een project zouden moeten gaan werken, hoe hoog de ontwikkelings- en onderhoudskosten zouden zijn. Om dat allemaal te weten komen, moet er een metriek bestaan, zoals we b.v. de meter voor de lengte hebben, die alle genoemde factoren kan uitdrukken in een vaste eenheid. Door de tijd heen heeft men geprobeerd om een geschikte manier voor het meten te vinden zoals b.v. kosten-per-defect of lines-of-code maar geen van deze methodes was betrouwbaar genoeg om een internationale standaard te worden.

Waarom is de lines-of-code metriek niet goed genoeg? Er kan namelijk niet vastgesteld worden of men fysieke of logische regels moet tellen of m.a.w. is een regel dat wat met "enter" eindigt of met een formeel scheidingsteken zoals punt of puntkomma. Verder moet men zich steeds afvragen of commentaar, lege regels, hergebruikte code ook als "lines-of-code" beschouwd mogen worden. Wat gebeurt er met zo'n metriek als we een applicatie hebben die geschreven is in meerdere programmeertalen? Kunnen we Ada en C++ applicaties met elkaar vergelijken als we de lines-of-code metriek hanteren? Schrijft elke programmeur hetzelfde aantal regels voor dezelfde opdracht? Op al deze vragen had de lines-of-code metriek geen adequaat antwoord.

De minpunten van metrieken zoals lines-of-code waren bekend aan IBM's werknemer Allan Albrecht die in oktober 1979 een nieuwe metriek publiceerde. Deze nieuwe techniek werd *"Functiepuntenmethode"* genoemd en werd snel de meest gebruikte manier om de omvang van software applicaties te meten. Albrecht baseerde zijn methode op de functies die aan eindgebruikers geleverd hoorden te zijn. Deze functies worden vastgesteld door de gebruikers tijdens de periode waarin de eisen voor een applicatie opgesteld worden en dan kunnen software engineers bepalen hoe groot een project zal worden en natuurlijk hoe hoog het budget ervoor moet zijn.

Maar de functiepuntenanalyse is pas eind jaren zeventig uitgevonden en voor die tijd is er een behoorlijk aantal applicaties ontworpen waarvan men ook graag de grootte en nodige onderhoudskosten zou willen weten. Daarvoor wil men natuurlijk ook het aantal functiepunten weten.

De ervaring leert dat de documentatie van programma's meestal niet consistent is en dat er naast de broncode die heel vaak erg onduidelijk is weinig informatie over reeds ontwikkelde applicaties beschikbaar is. Toch wil men het aantal functiepunten voor bestaande software kunnen meten en dat werk zou handmatig gedaan kunnen worden. Maar dat soort werk kost softwarebedrijven te veel tijd en geld en daarom wil men dat proces automatiseren.

Daarom bedachten A. Albrecht en C. Jones een nieuwe methode om aan het aantal functiepunten te komen op een redelijk simpele manier. Ze koppelden lines-of-code en de functiepuntenmetriek met elkaar en kregen een methode die *backfiring* genoemd werd. Voor elke programmeertaal werd een factor [2] gegeven waardoor het aantal regels broncode gedeeld wordt. Die factor is verkregen door het vergelijken van het aantal programmaregels met het aantal functiepunten in applicaties waarvoor ze over deze informatie beschikten. Verder is voor elke programmeertaal bepaald hoeveel regels assembler code er nodig zijn om een bepaald statement uit te drukken en op basis daarvan is er aan elke taal een taalniveau [2] toegekend. Met behulp van het niveau konden ze vaststellen welke factor bij welke taal behoort. Natuurlijk speelde ook de ervaring van Albrecht en Jones een grote rol in dat hele proces.

Maar wat gebeurt er als in een applicatie een andere programmeerstijl is gebruikt? Krijgen we dan met de bijbehorende factor weer een goed resultaat? Is één en dezelfde factor toepasbaar op zowel militaire projecten als op management informatiesysteem (MIS) projecten? Hoe nauwkeurig kan er vastgesteld worden wat een coderegel is? Hoe nauwkeurig is het eindresultaat van deze metriek?

Hoewel backfiring een snelle manier is om aan een aantal functiepunten te komen is het verkregen resultaat niet genoeg nauwkeurig (25% meetfout) om het meten daarop te kunnen baseren. Backfiring is afhankelijk van het getelde aantal regels (sommige bedrijven tellen fysieke en andere logische regels code - statements) en we hebben al gezien dat zo’n proces niet uniek is. Verschillende programmeerstijlen beïnvloeden deze methode doordat ze verschillende aantallen regels code produceren voor dezelfde functie.

Maar ondanks al zijn nadelen wordt backfiring gebruikt in veel tools voor het bepalen van software- en onderhoudskosten. De vraag is natuurlijk waarom een onbetrouwbare metriek dan gebruikt wordt. Het antwoord is heel eenvoudig. Er bestaat geen betere methode op de markt die in staat is om alleen op basis van de bestaande broncode het aantal functiepunten in een applicatie snel te bepalen.

Dat heeft ons aanleiding gegeven om te gaan onderzoeken of er een andere manier gevonden kan worden om een connectie tussen broncode en functiepunten te maken. Uitkomsten van de nieuwe methode zouden beter moeten zijn dan die van backfiring. Verder is ons doel om deze techniek volledig te automatiseren.

Voor een goed begrip van dit onderwerp bespreken we eerst theoretische aspecten van functiepuntenanalyse en daarna introduceren we onze nieuwe methode.

# Hoofdstuk 2

# Theoretische aspecten

## 2.1 Albrecht's functiepuntenmethode

Allan Albrecht werkte voor IBM's Data Processing Service Group. Hij kreeg opdracht om de productiviteit van een aantal projecten te meten. Daarbij analyseerde Albrecht 22 projecten waarvan 16 geschreven in COBOL, 4 in PL/1 en 2 in DMS/VS. Alle projecten waren volledig gedefinieerd van de eisenstelling tot aan de uiteindelijke systeemtests en demonstratie. Het totale aantal werkuren net zoals alle functionele factoren waren bekend. Op basis daarvan hebben Albrecht en zijn collega's een metriek ontwikkeld die de waarde van aan de eindgebruiker geleverde functies zou meten en daarnaast ook nog onafhankelijk was van de aanpak en de gebruikte technologie. Deze methode baseerde zich op het tellen van gebruikers invoeren, navragen, uitvoeren en project master files. Elk van de factoren werd vermenigvuldigd met een getal dat de functiewaarde voor de gebruiker uitdrukt.

Zo kregen ze:

Aantal invoeren x 4
Aantal uitvoeren x 5
Aantal navragen x 4
Aantal master files x 10

Som van deze getallen werd verder verhoogd of verlaagd met ±25% om de complexiteit uit te drukken. Uiteindelijk kregen ze dus een dimensieloos getal dat volgens Albrecht en zijn werkgroep effectief de functiewaarde uitdrukte die aan de klant geleverd was. En inderdaad was de methode van Albrecht de eerste in de geschiedenis van het meten van software waarmee de economische waarde van applicaties geschreven in hogere programmeertalen rechtstreeks gemeten kon worden. Maar toch had deze metriek haar nadelen. De twee problemen waren:
- de aanpassing met ±25% was niet goed genoeg voor real-time systemen
- het bepalen van de complexiteit van functionele factoren was helemaal subjectief.

De oplossing voor deze problemen kwam vijf jaar later en werd weer geleverd door IBM.

## 2.2 Herziene versie van de FP methode

In 1984 publiceerde Albrecht de herziene versie van zijn functiepuntenmethode uit 1979. De techniek had enkele belangrijke veranderingen ondergaan.

Ten eerste telden nu in plaats van vier, vijf functionele factoren mee in het meten van software. Dus naast invoeren, uitvoeren, navragen, en bestanden kwamen er ook interfaces bij. Verder werden deze factoren op basis van hun complexiteit gesorteerd als eenvoudige, gemiddelde en complexe elementen. Dat is het best te zien in de volgende tabel [2].

| Functionele parameter | Lage Complexiteit | Gemiddelde complexiteit | Hoge complexiteit |
|---|---|---|---|
| Externe invoer (IT) | ×3 | ×4 | ×6 |
| Externe uitvoer (OT) | ×4 | ×5 | ×7 |
| Externe navraag (QT) | ×3/4 | ×4/5 | ×6/7 |
| Intern logisch bestand (FT) | ×7 | ×10 | ×15 |
| Extern interfacebestand (EI) | ×5 | ×7 | ×10 |

Tabel 1: herziene versie van functiepuntenanalyse: alle functionele factoren en de complexiteitsmultiplicator

Om de complexiteit van de functionele factoren te bepalen werd er gekeken naar het aantal gebruikte data items en het aantal gerefereerde bestanden. Neem bij voorbeeld een functie die in een autokenteken database een nieuw kenteken invoert met de naam van de eigenaar van de auto. Dan hebben we dus een invoer die twee data items (kenteken en eigenaar) gebruikt en in de relatie met één bestand staat (kenteken database). Opgezocht in Tabel 2 zien we dat deze invoer eenvoudig is.

Een overzicht van alle elementen staat in de tabellen 2 t/m 6.

| Het aantal gerefereerde files | Het aantal gebruikte data 1-4 | 5-15 | 16 en meer |
|---|---|---|---|
| 0 of 1 | eenvoudig(3) | eenvoudig(3) | gemiddeld(4) |
| 2 | eenvoudig(3) | gemiddeld(4) | complex(6) |
| 3 en meer | gemiddeld(4) | complex(6) | complex(6) |

Tabel 2: invoeren gesorteerd op basis van de complexiteit

| Het aantal gerefereerde files | Het aantal gebruikte data 1-5 | 6-19 | 20 en meer |
|---|---|---|---|
| 0 of 1 | eenvoudig(4) | eenvoudig(4) | gemiddeld(5) |
| 2 of 3 | eenvoudig(4) | gemiddeld(5) | complex(7) |
| 4 en meer | gemiddeld(5) | complex(7) | complex(7) |

Tabel 3: uitvoeren gesorteerd op basis van de complexiteit

In de bovenstaande twee tabellen hebben we gezien hoe de complexiteit van invoeren en uitvoeren bepaald wordt. Omdat navragen een combinatie van deze twee zijn kijken we naar de complexiteit van zowel invoer als uitvoer (Tabel 4a en 4b) en kiezen het grootste getal uit.

***Invoer gedeelte***

| Het aantal gerefereerde files | Het aantal gebruikte data 1-4 | 5-15 | 16 en meer |
|---|---|---|---|
| 0 of 1 | eenvoudig(3) | eenvoudig(3) | gemiddeld(4) |
| 2 | eenvoudig(3) | gemiddeld(4) | complex(6) |
| 3 en meer | gemiddeld(4) | complex(6) | complex(6) |

Tabel 4a: de complexiteit van het invoer gedeelte van een navraag

***Uitvoer gedeelte***

| Het aantal gerefereerde files | Het aantal gebruikte data 1-5 | 6-19 | 20 en meer |
|---|---|---|---|
| 0 of 1 | eenvoudig(4) | eenvoudig(4) | gemiddeld(5) |
| 2 of 3 | eenvoudig(4) | gemiddeld(5) | complex(7) |
| 4 en meer | gemiddeld(5) | complex(7) | complex(7) |

Tabel 4b: de complexiteit van het uitvoer gedeelte van een navraag

| Het aantal gerefereerde files | Het aantal gebruikte data 1-19 | 20-50 | 51 en meer |
|---|---|---|---|
| 1 | eenvoudig(7) | eenvoudig(7) | gemiddeld(10) |
| 2 - 5 | eenvoudig(7) | gemiddeld(10) | complex(15) |
| 6 en meer | gemiddeld(10) | complex(15) | complex(15) |

Tabel 5: bestanden gesorteerd op basis van de complexiteit

| Het aantal gerefereerde files | Het aantal gebruikte data 1-19 | 20-50 | 51 en meer |
|---|---|---|---|
| 1 | eenvoudig(5) | eenvoudig(5) | gemiddeld(7) |
| 2 - 5 | eenvoudig(5) | gemiddeld(7) | complex(10) |
| 6 en meer | gemiddeld(7) | complex(10) | complex(10) |

Tabel 6: interfaces gesorteerd op basis van de complexiteit

Nu we hebben gezien op welke manier Albrecht de functionele elementen sorteerde gaan we kijken wat er verder in de herziene versie van functiepuntenmethode is gedaan.

De tweede stap in de verfijning van zijn functiepunten telling heeft Albrecht genomen door i.p.v. het verhogen of verlagen van het getelde aantal functiepunten met ±25% veertien complexiteitsfactoren in te voeren. Elk van deze factoren kan de waarde 0-5 krijgen en ze geven uiteindelijk een complexiteitsmultiplicator in het domein 0,65-1,35. De genoemde invloedsfactoren kijken naar:

1. ***Data communicatie***
   De taak van deze factor is om te beoordelen in hoeverre gegevens en/of kontrol informatie over een netwerk getransporteerd worden. Zo heeft data communicatie geen invloed op b.v. een batch applicatie en wordt er dan 0 aan deze factor toegekend maar voor een on-line systeem met overwegend teleprocessing heeft hij waarde 5.

2. ***Gedistribueerde functies***
   Als een applicatie uitgevoerd wordt op één processor is de waarde van deze factor 0 en naarmate de hoeveelheid processoren waarop een applicatie uitgevoerd wordt stijgt, neemt de waarde van deze factor ook toe.

3. ***Uitvoering van de applicatie***
   Als er geen speciale uitvoeringseisen zijn gesteld voor een applicatie, heeft deze factor geen impact op het systeem. Maar hoe specifieker ontwerp, ontwikkeling en implementatie eisen zijn (zoals b.v. snelle reacties van het systeem), hoe meer tools voor de analyse van design, ontwerp enz. gebruikt moeten worden en dat verhoogt de waarde van deze factor.

4. ***Moeilijkheidsgraad van de configuratie***
   Als een applicatie uitgevoerd wordt op standaard apparatuur en er bestaan geen restricties voor gebruik van hardware is deze factor ook niet van belang voor de functiepunten analyse. Terwijl het toepassen van de FP methode moeilijker wordt als er verschillende gebruiksbeperkingen zoals b.v. het minimaliseren van swap segmenten opgelegd worden voor een applicatie die op en gedistribueerde systeemconfiguratie uitgevoerd wordt.

5. ***Transactiegraad***
   Afhankelijk van de hoeveelheid transacties die een applicatie moet gaan uitvoeren wordt de moeilijkheidsgraad van de uitvoeringsanalyse bepaald. Hoe ingewikkelder deze analyse is hoe hoger de waarde van deze complexiteitsfactor wordt.

6. ***On-line invoer van gegevens***
   Als er minder dan 15% van de transacties aan interactieve data invoer besteedt wordt is de waarde van deze factor minder dan 3. In het geval on-line invoer tussen 30% en 50% van de transacties in beslag neemt, wordt aan deze factor de waarde 5 toegekend.

7. ***Efficiëntie van het ontwerp voor eindgebruikers***
   Afhankelijk van de moeilijkheidsgraad van gebruikerseisen wordt de invloed van deze factor op het systeem bepaald. Standaard ontwerp heeft weinig impact op de applicatie terwijl men voor een applicatie waarvoor speciale efficiëntie-eisen gesteld worden meer tijd en kennis nodig heeft en dat weerspiegelt zich in het aantal functiepunten.

8. ***On-line bewerkingen***
   Als er voor applicaties waarin on-line bewerking van gegevens plaatsvindt speciale maatregelen genomen moeten worden zoals b.v. het voorkomen van verlies van gegevens wordt de implementatie van het systeem ingewikkelder en daarmee wordt de waarde van deze factor hoger. Aan de andere kant is dit punt niet van toepassing indien er geen on-line bewerkingen in een applicatie zijn.

9. ***Complexiteit van het proces***
   De naam van deze factor zegt al veel. Voor processen waarvoor complexe invoer , uitvoer, wiskundige berekeningen e.d. nodig zijn kennen we een hogere waarde aan deze factor toe dan voor een proces waarvoor geen extra inspanning nodig is om bepaalde taken te verrichten.

10. ***Geschiktheid voor hergebruik van de code***
    Deze factor scoort een 0 als een applicatie voor maar één doel is gemaakt terwijl de factorwaarde naar 5 stijgt als de applicatie in zijn geheel hergebruikt moet worden.

11. ***Gemak van de installatie en conversie***
    Als de gebruiker geen speciale installatie en conversie eisen stelt, heeft deze factor geen invloed op de functiepuntenanalyse. Maar als er voor deze taken handleidingen of zelfs tools ontworpen moeten worden, dan krijgt deze factor een hogere waarde.

12. ***Operationeel gemak***
    Hoe meer een applicatie gebruikersvriendelijk moet zijn hoe meer inspanning er nodig is voor zijn ontwikkeling en dat betekent dus ook een hogere invloed van deze factor op het aantal functiepunten.

13. ***Gebruik door meerdere sites***
    Hetzelfde geldt natuurlijk voor een applicatie die door meerdere locaties tegelijk gebruikt wordt.

14. ***Gemak in het aanbrengen van veranderingen***
    Als een applicatie in de toekomst uitgebreid of veranderd moet worden en de ontwerpers een mogelijkheid daarvoor moeten inbouwen, dan wordt de ontwikkeling van zo'n applicatie moeilijker en daarmee wordt de waarde van deze factor ook hoger.

Als men alle factoren heeft bepaald kan het aantal functiepunten nog nauwkeuriger uitgerekend worden met behulp van de volgende formule:

FP totaal = totaal aantal getelde FP x
(0,65 + 1% van de som(S) van alle 14 complexiteitsfactoren)

Uit deze formule zien we ook dat de complexiteitsmultiplicator inderdaad tussen 0,65(S=0) en 1,35(S=70) ligt

We weten nu wat de functionele factoren voor het tellen van functiepunten zijn en hoe hun complexiteit bepaald kan worden. We zullen nu uitleggen waar en hoe we deze factoren kunnen vinden en we zullen bepalen of ze wel of niet een rol spelen in de functiepuntenanalyse.

## 2.3 Invoer

Elke unieke gebruikers data of kontrol invoer die in een applicatie veranderingen in interne bestanden, data sets of tabellen veroorzaakt, wordt als invoer beschouwd. Hieronder vallen niet alleen gegevens die direct door de gebruiker ingevoerd worden maar ook invoer bestanden en transacties ontvangen van een andere applicatie.
Een invoer is uniek als:
1. hij een uniek formaat heeft of
2. hij wel hetzelfde formaat heeft als een andere invoer maar andere proceslogica nodig heeft voor het verwerken van invoergegevens.

Uniek formaat betekent dat het tuple (data items, aantal refererende bestanden) uniek is maar de term proceslogica vraagt om wat meer uitleg. We gaan weer terug naar het voorbeeld van de kenteken database. Stel dat een auto verkocht wordt waarvan de naam van de eigenaar en het kenteken al ingevoerd zijn in de database. Dan moet er een invoerfunctie bestaan die deze verandering kan registreren. Deze functie zou het kenteken en de naam van de nieuwe eigenaar moeten invoeren. Dus we krijgen een functie met hetzelfde formaat als de functie die een nieuwe auto in de database invoert. Maar de ene functie voert de gegevens in de database in en de andere bewerkt al bestaande databaserecord dus verricht een ander soort verandering in de database. Dat betekent dat we met twee verschillende invoeren te maken hebben hoewel hun formaat hetzelfde is. Deze twee functies hebben dus verschillende proceslogica.

Nu kunnen we onderscheid maken tussen verschillende invoeren maar waar moeten we naar invoeren gaan zoeken en hoe moeten we ze tellen?

Tel als **één** invoer elk uniek geformatteerd of verwerkt deel van:
- Wachtwoord of verzameling conversationele schermen om toegang tot de applicatie te krijgen.
- Scherminvoer (tel bevestigingsboodschap als één data item).
- Functie invoer via het scherm (één per functie – add, change, delete).
- Data set of master[1] file (drie per file of data set).
- Nieuwe master file ontstaan na het sequentiële bijwerkingproces – gegeven een oude master file (al geteld) om bijgewerkt te worden door een transactie tape (telt als of één invoer of één interface). De nieuwe master file zal geteld worden als een nieuw intern bestand en mogelijk ook als een interface als het geëxporteerd moet worden.
- Navraag gevolgd door update invoer (plus één navraag).
- OCR scanregel.
- Bar code lezer.
- Andere door machine leesbare invoervormen.
- Credit kaart, ATM kaart, beveiligingspas, magnetische invoer enz.
- Twee invoerschermen elk met hetzelfde formaat maar verschillende proceslogica – tel één invoer per scherm (d.w.z. twee invoeren).
- Automatische data of transactiefile[2] (plus één interface als dataconversie is geëist). Één invoer wordt geteld voor elke transactie met verschillend type binnen een bestand. Als invoerbestanden aan elkaar geplakt worden (appended of joined) en voor elke file verschillende proceslogica vereist is, tel voor elk betrokkenen fileformaat één invoer.
- Door de gebruiker onderhouden tabel of bestand – meestal drie invoeren, één uitvoer en één navraag.

[1] voor meer uitleg over de term *master file* zie paragraaf 2.6
[2] voor meer uitleg over de term *transactiefile* zie paragraaf 2.6

- Data, functie en kontrol transactie van andere applicaties (plus één interface).
- Selectie menu invoer met '*save*' mogelijkheid (plus één navraag).
- Kontrol informatie voor rapport selectie.
- Gebruikers kontrol invoer - sign on, sign off, wachtwoord, switch settings – elk veld is één data item.
- Invoer uitgedrukt in een querytaal tenzij een navraag – één per invoer.
- Updatefunctie die een onderdeel is van een rapport of query voorziening.

Aan de andere kant zijn er ook acties die niet als een invoer beschouwd mogen worden en dat zijn:
- Automatische invoer van andere programma's uitgevoerd door het systeem of automatische inschakeling van beveiligingsprocedures die meestal in mainframe plaatsvindt.
- Het invoeren van een master file. Dit wordt als een interface geteld.
- Individuele boodschappen zoals informatie, bevestiging, foutmelding enz.
- Invoer gedeelte van een navraag.
- Menuscherm.
- Backup invoer die hetzelfde formaat en proceslogica heeft als primaire invoer.

## 2.4 Uitvoer

Uitvoer is elke uniek gegenereerde data of kontrol uitvoer die de applicatiegrens verlaat. Hieronder vallen niet alleen rapporten en boodschappen verzonden naar de applicatie gebruiker via scherm en printer maar ook naar andere applicaties en hun gebruikers.
Een uitvoer is uniek als:
1. hij een uniek formaat heeft of
2. hij wel hetzelfde formaat heeft als een andere uitvoer maar andere proceslogica (b.v. twee uitvoeren met namen van de eigenaar en het kenteken maar één verkregen uit kentekendatabase en de andere uit wegenbelasting database worden als twee uitvoeren geteld)

Tel als **één** uitvoer elk uniek geformatteerd of verwerkt deel van:
- Geformatteerde uitdraai van master of transactiefile.
- Backup file.
- Ad-hoc rapport generator of DBMS rapport writer sjabloon.
- Scherm uitvoer - een per uniek formaat, detail, proceslogica enz.
- Batch geprinte uitvoer - hetzelfde als voor schermen.
- Batch run kontrol rapport.
- Batch error rapport.
- Single-level sortering met verschillende sleutel, alleen als herformatteren nodig is - b.v. er is een kopie van een rapport gemaakt en daarin zijn bepaalde gegevens gewist. Het nieuwe (gekopieerde) rapport wordt in andere functie gebruikt en daar heeft het ook een ander doel dus telt het als een andere uitvoer.
- Elke verschillende volgorde in verschillende rapporten van multi-level filesortering als hij geformatteerd is.
- Elk verschillend uitgerekend totaal in een rapport - kolom, groep, minor, major, grand enz.

- Elke verschillende bewerkingsperiode van een rapport - dagelijks, wekelijks, maandelijks, jaarlijks enz.
- Kontrol totalen rapport.
- Kontrol of checklist rapport.
- Check register.
- Uitvoer van het selectie menuscherm met "*save*" mogelijkheid.
- Start scherm uitvoer - zoals de initialisatie van conversationele bewerking.
- Ending scherm uitvoer - zoals EOJ(end-of-job) bevestiging.
- OCR scanregels.
- Detail scherm uitvoer.
- Gebruikerstoegankelijke rapport file.
- Door gebruiker onderhouden tabel of file - meer dan één uitvoer mogelijk en meestal drie invoeren en één navraag.
- Boodschappen naar operator.
- Gegevens, functie of kontrol transactie naar een andere applicatie.
- Uitvoer bestand van automatische gegevens of bewerkte transactie records verzonden naar een andere applicatie (plus één interface).
- Query taal uitvoer tenzij een navraag - één per uitvoer.
- Rapportfile verzonden naar een andere applicatie voor het printen, formulieren, grafiek (plus één interface voor het ontvangende programma).

Maar de volgende worden niet als een uitvoer beschouwd:

- Koppen, paginanummers, kolomnamen enz.
- Herhaalschermen, gedrukte pagina's of detailregels.
- Uitvoer van een selectiemenu.
- Uitvoer gedeeltes van een navraag.
- Uitvoerbestanden.

## 2.5 Navragen

Elke unieke invoer/uitvoer combinatie waarbij een on-line gedefinieerde invoer een onmiddellijke uitvoer genereert, word een navraag (inquiry) genoemd. Een navraag is uniek als:

1. of zijn invoer- of uitvoergedeelte een uniek formaat hebben of
2. indien het formaat hetzelfde is maar verschillende proceslogica wordt vereist in een van de twee gedeeltes van een navraag.

Maar lang niet elke invoer/uitvoer combinatie wordt als een navraag beschouwd. Als er b.v. een record tijdens het invoer/uitvoer proces bijgewerkt wordt, dan is er geen sprake van een navraag. Men telt dan gewoon een invoer en een uitvoer. Om er achter te komen of we met een navraag te maken hebben, is er een definitie opgesteld en die luidt als volgt:

Een invoer-uitvoer koppel is een navraag als

1. Gebruik van **eenvoudige sleutels** is gemaakt om data te extraheren – een enkel record of een groep gerelateerde records.
2. **Beantwoording** op de invoeractie **onmiddellijk** plaats vindt.
3. Er **geen bewerkingen** verricht worden.

Als een van de bovengenoemde regels niet vervuld is tellen we *een (of meer) invoeren* en *een (of meer) uitvoeren* maar GEEN navraag.

Tel als **één** navraag elk uniek geformatteerd of verwerkt deel van de volgende (als dat ook voldoet aan de regels 1 – 3):
- On-line invoer en uitvoer zonder update.
- On-line navraag in een transactie file die de applicatiegrens verlaat.
- Navraag gevolgd door een update invoer (plus één invoer).
- Hulpscherm of een informatieboodschap.
- Selectie menuscherm invoer en uitvoer (complete verzameling is één navraag).
- Selectie menuscherm invoer met save mogelijkheid (plus één invoer).
- Door de gebruiker onderhouden tabel of bestand.
- Navraag gedeeltes van een query voorziening of query taal.

Net als bij de invoer en uitvoer zijn er bij de navraag ook de situaties waarbij je geen navraag mag tellen en dat zijn:
- Navragen die niet aan de condities 1-3 voldoen.
- Transfer van een commandoregel van en/of naar een ander scherm.
- Navragen waarbij of het invoer- of het uitvoergedeelte niet aan de regels voor respectievelijk invoer en/of uitvoer voldoen.

## 2.6 Bestanden

Elke logische groep gebruikersdata of kontrol informatie geheel onderhouden **binnen** de applicatiegrens wordt als een bestand beschouwd. In de functiepuntenanalyse wordt een verschil gemaakt tussen transactie files en permanente files. Met een transactiebestand worden gegevens van en/of naar een applicatie getransporteerd en verder bevat zo'n bestand geen gegevens waarmee gebruikers 'gediend' kunnen worden. Aan de ander kant is de inhoud van een permanent bestand van cruciaal belang voor het uitvoeren van gebruikersfuncties. Een voorbeeld van een permanent bestand is onze kenteken database waar alle gegevens over auto's en hun eigenaren in zitten. Een transactie bestand zou een bestand kunnen zijn dat b.v. alle kentekens en de eigenaren bevat die vandaag in Amsterdam geregistreerd zijn. Zo'n bestand wordt dan elke werkdag om 18.00 toegevoegd aan de kenteken database en in dat geval is dat record geen bestand maar een invoer.

Het onderscheid tussen permanente bestanden en transactiebestanden wordt gemaakt omdat alleen **permanente files** als logische bestanden beschouwd worden. Deze krijgen namen zoals: "master file", "updated records", "refererend bestand", "vaste data", "status", "record", of "historische files". Als deze ook door een andere applicatie gebruikt worden, tellen ze voor zowel interfaces als permanente files.

Tel als **één** intern logisch bestand elke logische groep gebruikersgegevens of kontrol informatie binnen de applicatiegrens:
- Master file die bijgewerkt wordt.
- Logische verzameling gegevens gezien vanuit gebruikersinvalshoek.
- Interne database.

- Database logische gebruikersviews – één per view. Joins in een relationele database worden alleen geteld als ze noodzakelijk zijn om aan een goedgekeurde gebruikerseis te voldoen.
- Data base toegangspad binnen views - één per pad.
- Logische interne files gegenereerd of onderhouden door de applicatie.
- Files of datasets toegankelijk voor gebruikers via wachtwoorden, sleutelwoorden of parameters.
- Files of datasets verzonden naar of samen met een andere applicatie gebruikt (plus één interface).
- Door de gebruiker onderhouden tabel of file (plus drie invoeren, één uitvoer en één navraag).
- Sequentiële batch applicatie.
- Elk bestand dat aan een andere file toegevoegd wordt en daarbij ook verschillende proceslogica heeft.
- Nieuwe master file ontstaan na een sequentieel updateproces – de oude master file wordt ook geteld.

Maar er worden niet geteld:
- Interne logische bestanden waarvan maar een functie extern is – b.v. backup files.
- Bestanden die onderhouden worden door een andere applicatie.
- Hard-gecodeerde informatie.
- Transactiefiles of printfiles.
- Interne systeembestanden, tijdelijke sorteerbestanden e.d.
- Datastructuren gecreëerd met behulp van het COBOL "REDEFINES"[3] statement.

## 2.7 Interfaces

Elke logische file of andere logische groep gegevens of kontrol informatie binnen de applicatiegrens die wordt verzonden naar, gedeeld met of ontvangen van een andere applicatie wordt als een interface beschouwd. Bestanden die tussen de applicaties gedeeld worden en door alle applicaties gebruikt worden tellen als zowel interfaces als files voor beide applicaties. De applicatie die het werk op een bestand verricht (bewerken, onderhouden enz.) mag *een file* tellen maar alle applicaties tellen wel *een interface per applicatie*. Dwz. elke interfacefile moet een interne file zijn van een (of meer) in het proces betrokkene applicaties of het moet een transactie of printfile zijn.

Tel als **één** interface elke logische groep gegevens of kontrol informatie binnen de applicatie die of wordt verzonden naar of gedeeld met of ontvangen van een andere applicatie:
- File of records *van* een andere applicatie - voor andere applicatie – één file en één interface.
- File of records *naar* een andere applicatie (plus één file) - voor andere applicatie één interface.
- File of records *naar* andere applicaties (plus één file).

[3] M.b.v. deze clausule kan men meerdere beschrijvingen geven van één geheugengebied (veld, tabel e.d.) terwijl de geheugenpositie van herbenoemde datastructuur onveranderd blijft

- File of records *gedeeld tussen* twee of meer applicaties (plus één file) – voor andere applicaties plus een interface en als een applicatie onderhouden wordt dan nog plus één file.
- Data base *gedeeld met* andere applicaties (plus één file) , ÉÉN interface per verzonden view - voor andere applicatie één file plus één interface per view.
- Transactiefile *ontvangen van* andere applicatie **met vereiste data conversie** is één interface (plus één invoer).
- Transactie file *verzonden naar* een andere applicatie **met vereiste data conversie** is één interface (plus één uitvoer).
- Parameter lijst of gebruikerscode/wachtwoord verzonden naar of ontvangen van een andere applicatie.

MERK OP dat het delen van een master file aan elke applicatie zowel een file als een interface toekent.

Net als bij de voorgaande vier FP elementen kent het tellen van interfaces situaties waarin geen interfaces geteld worden:
- Als de logische bestanden binnen één applicatie gedeeld worden.
- Logische bestand dat niet aangevraagd wordt maar toch een fysiek deel van een ander bestand uitmaakt.
- Transactiebestanden waarvoor geen data conversie vereist wordt (zo'n bestand is of een invoer of een uitvoer).
- Bestanden die alleen uit boodschappen bestaan.

## 2.8 Voor- en nadelen van de functiepuntenmethode

In deze tekst is al eerder gezegd dat de functiepuntenmetriek van groot belang is voor het uitdrukken van de economische waarde van software op een objectieve en consistente manier. Maar naast deze positieve eigenschap zijn er nog een paar. Zoals:
- Één uniek manier om alle applicatietypes te meten.
- Het normaliseren van gegevens wat een mogelijkheid biedt om projecten met elkaar te vergelijken.
- Het representeren van meetresultaten in een vorm die voor mensen buiten de IT branche ook begrijpelijk is wat meteen bijdraagt aan de verbetering in de communicatie tussen technici en klanten.
- De functiepuntenmetriek is heel gemakkelijk te begrijpen en toe te passen door zowel software engineers als klanten.
- De grootte van een applicatie is in een vroeg ontwikkelingstadium te bepalen.

Toch is deze techniek niet perfect en met bovenstaande voordelen gaan ook enkele nadelen gepaard. Zo valt het heel makkelijk te ontdekken dat de functiepuntenmethode niet gemakkelijk geautomatiseerd kan worden en dat er soms toch subjectieve beslissingen genomen moeten worden b.v. tijdens het bepalen van de waarden van veertien complexiteitsfactoren. Verder kan het moeilijk zijn om er achter te komen tot welke categorie een component behoort. Het opnemen van een klein systeem in een groter levert minder functiepunten op dan de som van het aantal functiepunten van beide systemen terwijl die twee getallen hetzelfde horen te zijn. Mensen die functiepunten moeten gaan tellen hebben een speciale opleiding nodig om consistente resultaten van tellingen te krijgen. Dat betekent voor een bedrijf dat er extra geld besteedt moet worden aan cursussen. Dat laatste wordt meestal als een nadeel gezien hoewel dat in feite een voordeel is voor de werkgever. Een onderzoek [15] heeft bewezen dat een werknemer significant productiever is als hij maar tien werkdagendagen per jaar aan een opleiding of een cursus besteedt.

## 2.9 Verdere ontwikkelingen

De functiepuntenmetriek is natuurlijk in de twintig jaar van zijn bestaan niet de enige techniek gebleven die kwaliteit en grootte van software mat. In de loop der jaren zijn de methodes zoals De Marco's "Bang" metriek, Feature Point, Mark II functiepuntenmetriek, backfiring enz. ontwikkeld maar toch is de functiepuntenmethode tot nu toe een van de meest gebruikte op de markt. Bovendien was de vraag naar deze techniek zo groot dat in 1986 een non-profit organisatie, International Function Point Users Group (IFPUG), ontstond. IFPUG heeft standaarden opgesteld voor het tellen van functiepunten en daarmee geholpen bij het creëren van een unieke metriek. Verder doet deze organisatie veel aan de ontwikkeling van allerlei metrieken en de verbetering van de functiepuntenanalyse. IFPUG heeft onder andere een handleiding voor het tellen van functiepunten uitgegeven. Tegenwoordig heeft IFPUG 1200 leden in meer dan 30 landen.

Naast deze organisatie zijn wereldwijd ook de landelijke functiepunten associaties opgericht en één daarvan is Nederlandse Software Metrieken Gebruikers Associatie (NESMA). Momenteel heeft de NESMA ongeveer 130 leden in Nederland en is de grootste vereniging in zijn soort in Europa en is de op één na grootste ter wereld.

Een interessant feit is dat de telregels van de IFPUG en de NESMA verschillen in bepaalde punten en de resultaten die  de tellingen volgens de ene of de andere methode opleveren, kunnen voor een en hetzelfde systeem verschillend zijn.

# Hoofdstuk 3

# Methodiek

In het voorafgaande hoofdstuk zijn de theoretische aspecten van functiepuntenanalyse besproken. Maar het doel van dit onderzoek is zoals we al weten het toepassen van de functiepuntenmethode op broncode van bedrijfsapplicaties. Om dat te bereiken is het noodzakelijk om functionele factoren: invoer, uitvoer, navraag, bestanden en interfaces op te sporen in applicatieprogramma's. Dit houdt in dat we in een programmeertaal op zoek moeten gaan naar de statements die deze functionele factoren implementeren.

De programmeertaal bestudeerd in dit onderzoek is ANS-COBOL. Misschien lijkt de taalkeuze een beetje vreemd maar de functiepuntenmethode gebruikte in haar ontwikkeling de MIS applicaties omdat er een grote behoefte ontstond om de omvang van dat soort systemen te weten (voor de berekening van b.v. onderhoudskosten). En het feit dat negentig procent van de MIS systemen in COBOL is geschreven, verklaart dan de taalkeuze. Naast COBOL wordt in het onderzoek ook SQL beschouwd omdat in COBOL programma's regelmatig gebruik wordt gemaakt van deze query taal. Voor on-line design en implementatie van COBOL systemen wordt vaak IBM Customer Information Control System (CICS) gebruikt en dat hebben we in dit onderzoek ook bestudeerd.

## 3.1 Functiepunten en COBOL

Na het bestuderen van COBOL, SQL en CICS statements kon er een verband gelegd worden tussen deze programmeertalen en functiepunten. Die connectie is gepresenteerd in de Tabel 7a-c.

Tijdens het toepassen van de FP methode op de programmeertalen hadden we moeite met het onderscheiden van transactie en master files omdat we uit de broncode niet kunnen afleiden of een bestand permanent of tijdelijk is. Er moet namelijk onderzocht worden wat voor soort werk een bepaald programma verricht in een applicatie en op basis daarvan moet men zelf concluderen of er sprake is van een permanente file of een transactie file. Verder kwamen we in de problemen met het bepalen van de applicatiegrens. Want functiepunten-analyse behandelt alleen gebruikersfuncties die binnen de applicatiegrens zitten maar we konden vanuit de broncode niet afleiden waar die grens ligt. Er is toen besloten dat er gekeken moet worden naar de relaties tussen de programma's in een applicatie en we hebben aan de eigenaar van de broncode gevraagd om de code per applicatie te leveren.

In de tabellen 7a, 7b en 7c staan respectievelijk COBOL, SQL en CICS statements opgesomd die van belang zijn voor functiepuntenanalyse.

| INVOER | UITVOER | NAVRAAG | BESTAND | INTERFACE |
|---|---|---|---|---|
| ACCEPT | DISPLAY(S) | DISPLAY(S) gevolgd door ACCEPT | SEARCH | CALL |
| SORT met USING | SORT met GIVING | SEARCH | CALL | OPEN INPUT |
| MERGE met USING | MERGE met GIVING | | SORT met GIVING | OPEN OUTPUT |
| SEARCH | SEARCH | | MERGE met GIVING | OPEN I-O |
| OPEN INPUT | OPEN OUTPUT | | OPEN INPUT | OPEN EXTEND |
| OPEN I-O | OPEN I-O | | OPEN OUTPUT | |
| CALL | OPEN EXTEND | | OPEN I-O | |
| | CALL | | OPEN EXTEND | |

Tabel 7a: COBOL statements die nodig zijn om functiepunten te tellen

| INVOER | UITVOER | NAVRAAG | BESTAND | INTERFACE |
|---|---|---|---|---|
| CREATE TABLE | CREATE TABLE | CREATE TABLE | CREATE TABLE | SET CONNECTION |
| INSERT | | SELECT | CREATE VIEW | INCLUDE |
| DELETE | | | SET CONNECTION | |
| UPDATE | | | INCLUDE | |

Tabel 7b: SQL statements die nodig zijn om functiepunten te tellen

| INVOER | UITVOER | NAVRAAG | BESTAND | INTERFACE |
|---|---|---|---|---|
| RECEIVE (MAP) | SEND | CONVERSE | SET FILE OPEN | SET FILE OPEN |
| SIGN(ON/OFF) | SEND TEXT | | CALL | CALL |
| DELETE | SEND PAGE | | LINK | LINK |
| (RE)WRITE | SEND MAP | | XCTL | XCTL |
| SET FILE OPEN | JOURNAL | | | ALLOCATE |
| CALL | SET FILE OPEN | | | CONNECT PROCESS |
| LINK | CALL | | | |
| XCTL | LINK | | | |
| | XCTL | | | |

Tabel 7c: CICS statements die nodig zijn om functiepunten te tellen

Waarschijnlijk valt het meteen op dat bepaalde statements in meerdere kolommen voorkomen. De oorzaak daarvoor is het indelen van bestanden naar permanente en transactiebestanden. We gaan nu alle kolommen doorlopen en leggen uit waarom onze keus op deze statements is gevallen.

## 3.2 Kolom "Invoer"

- ACCEPT - Dit statement leest eenvoudig invoer van het scherm en het wordt natuurlijk als de invoer gezien door de FP methode. Maar ACCEPT heeft ook een optie die van het systeem datum, tijd, dag e.d. opvraagt en onze methode ziet dat niet als een invoer. Dus als een ACCEPT statement in COBOL code voorkomt wordt hij als invoer beschouwd alleen als ermee gegevens van het scherm opgehaald worden. De volgende uitzondering op deze regel is als we een ACCEPT statement hebben die op een of meer DISPLAY statements volgt. Dan tellen we deze combinatie als een navraag en niet als invoer en uitvoer apart.
- SORT met USING - Zoals zijn naam al zegt, worden er met dit statement een of meer bestanden gesorteerd. Parameters van de USING optie zijn bestanden die gesorteerd moeten worden. Die worden automatisch geopend en dus niet met een OPEN INPUT statement. Omdat deze files alleen gegevens voor verdere processing leveren, worden ze als transactie files beschouwd en dus worden ze alleen als invoer gezien.
- MERGE met USING - Doet hetzelfde als SORT alleen moeten er in de USING optie minstens twee bestanden worden aangeroepen.

| | | |
|---|---|---|
| - SEARCH | - | Zoekt in een tabel die gedefinieerd wordt in COBOL broncode. In het theoretische gedeelte hebben we al gezegd dat er voor een door de applicatie of gebruiker gecreëerde en onderhouden tabel drie invoeren geteld moeten worden. Maar bij dit statement moet men er zeker van zijn dat de gebruikte tabel niet m.b.v. het REDEFINES statement is ontstaan. Als dat wel het geval is wordt het SEARCH statement niet geteld. |
| - OPEN INPUT | - | Opent een bestand voor het invoeren van gegevens. Alleen in het geval het om een transactiefile gaat, wordt dit statement als een invoer beschouwd en anders tellen we het als een bestand. Er moet verder ook gekeken worden of het geopende bestand gelezen wordt voordat het gesloten wordt. Want een bestand dat niet gebruikt wordt, speelt in functiepuntenanalyse geen rol. |
| - OPEN I-O | - | Zie voorgaand statement. |
| - CALL | - | Roept een programma aan. Dit statement wordt als een invoer beschouwd als het vanuit een transactieprogramma een ander programma aanroept. Als de gegevens die getransporteerd worden nog een conversie moeten ondergaan dan wordt er gekeken welk programma converteert en dat programma telt dan nog een interface. In het geval het aangeroepen programma binnen de applicatie is, wordt CALL niet gerekend als een functiepunt. |
| - CREATE TABLE | - | Is een SQL statement voor het aanmaken van database tabellen. Zoals in het geval van de SEARCH wordt een gecreëerde tabel beloond met drie invoer functiepunten. |
| - INSERT | - | Voegt een record toe aan een database tabel en daarvoor wordt een invoer geteld. |
| - DELETE | - | Hetzelfde geldt voor de verwijdering van een record uit een DB tabel. |
| - UPDATE | - | We tellen een invoer functiepunt als een database record bewerkt wordt. |
| - RECEIVE (MAP) | - | Gegevens die een CICS kontrol programma van het scherm heeft gelezen worden in een COBOL programma opgenomen en voor deze actie telt men een invoer functiepunt. |
| - SIGNON | - | Log in op een terminal is een invoer functiepunt waard. |
| - SIGNOFF | - | Voor het uitloggen van de genoemde terminal geldt hetzelfde. |
| - DELETE | - | Een bestandsrecord word verwijderd en daarvoor worden dezelfde regels als voor het soortgelijke SQL statement gehanteerd. |
| - REWRITE | - | Dit CICS statement werkt als SQL's UPDATE. |
| - WRITE | - | Een nieuw record wordt aan een bestand toegevoegd en dit commando beschouwt men als SQL's INSERT. |
| - SET FILE OPEN | - | Verricht hetzelfde werk als OPEN in COBOL. |
| - CALL en<br>- LINK | - | Werken als CALL in COBOL. |
| - XCTL | - | Doet hetzelfde als de vorige twee statements maar met het XCTL statement wordt de kontrol van transacties gegeven aan het aangeroepen programma. |

## 3.3 Kolom "Uitvoer"

| | |
|---|---|
| - DISPLAY | - Schrijft een boodschap op het scherm. Maar voor de FP methode zijn we alleen geïnteresseerd in aaneengesloten blokken van DISPLAY statements. Want als meer DISPLAY regels in een programma achter elkaar komen betekent dat meestal dat er één samenhangend geheel uitgevoerd wordt naar het scherm. Zulke blokken worden als uitvoer beschouwd indien ze niet door een ACCEPT zijn gevolgd. |
| - SORT met GIVING en - MERGE met GIVING | - In de voorafgaande paragraaf is de functie van deze twee statements beschreven. Als men de optie GIVING gebruikt, betekent het dat het resultaat van de sorteeractie in een bestand opgeslagen gaat worden. Dit bestand wordt eerder in het programma niet geopend voor schrijven dus moeten we het nu als een uitvoer rekenen. Hierbij moeten we ook rekening houden met het filesoort. Als dat een tijdelijk bestand is dan tellen we een uitvoer functiepunt. Maar als het om een master file gaat dan moet het als een bestand beschouwd worden. |
| - SEARCH | - Omdat er voor een door de applicatie of gebruiker gemaakte tabel een uitvoer functiepunt gerekend moet worden, hebben we dit statement ook in de kolom 'uitvoer' opgenomen. Maar let goed op dat de tabel niet met REDEFINES is gedefinieerd. |
| - OPEN OUTPUT en - OPEN I-O | - In een bestand dat voor uitvoer of I-O geopend is, moet er geschreven worden zodat het als een functioneel element gezien wordt door de FP methode. Als het om een transactiefile gaat dan tellen we een uitvoer functiepunt en anders telt het als een bestand. |
| - OPEN EXTEND | - Een bestand wordt met EXTEND geopend als men achter de bestaande gegevens nog andere records wil toevoegen. De bedoeling is dat er in dat bestand geschreven wordt en als dat echt gedaan is en het bestand geen master file is dan hebben we nog een uitvoer functiepunt te tellen. Indien het bestand wel een master file is dan is hij een bestand functiepunt waard. |
| - CALL | - De applicatie waar het programma aangeroepen uit een transactiefile zich bevind krijgt een uitvoer functiepunt credit. In geval dat programma ook data conversie moet verrichten mogen we nog een interface daarbij optellen. |
| - CREATE TABLE | - Voor dit statement geldt dezelfde logica als voor SEARCH (zie boven). |
| - SEND (MAP/ PAGE/ TEXT) | - Het CICS statement om een (on)geformatteerde boodschap naar het scherm te sturen. Doet hetzelfde als een of meerdere DISPLAY statements. |
| - JOURNAL | - Gegevens voor het journaal uitvoer bestand worden gezet in de journaal uitvoer buffer. |
| - SET FILE OPEN | - Zie vorige paragraaf. |
| - CALL en - LINK | - Zie vorige paragraaf. |
| - XCTL | - Zie vorige paragraaf. |

## 3.4 Kolom "Navraag"

| | |
|---|---|
| - DISPLAY gevolgd door ACCEPT | - Zoals al in paragrafen 3.2 en 3.3 is gezegd wordt een combinatie van een DISPLAY blok en een ACCEPT statement als een navraag beschouwd. |
| - SEARCH | - Hiervoor geldt hetzelfde als voor het SEARCH statement uit vorige paragraaf. |
| - CREATE TABLE | - Dit statement hanteert dezelfde regels als SEARCH behalve dat er geen mogelijkheid bestaat om een DB tabel te herdefiniëren met het REDEFINES statement. |
| - SELECT | - Vraagt de gegevens opgeslagen in een database tabel op. Als de opgevraagde data via een eenvoudige sleutel verkregen worden dan beschouwen we de actie van vraag en antwoord als een navraag. |
| - CONVERSE | - Wordt gebruikt voor communicatie tussen processen. In dat geval verwacht het zendende proces een antwoord van het ontvangende proces en dat ziet de FP methode als een navraag. |

## 3.5 Kolom "Bestand"

| | |
|---|---|
| - SEARCH | - Voor dit statement geldt hetzelfde als voor het SEARCH in de vorige twee paragrafen. |
| - CALL | - Als een transactie file een CALL doet naar een ander programma dan rekent de zendende applicatie één bestands functiepunt. |
| - SORT met GIVING en - MERGE met GIVING | - Als het resultaat van de sorteeractie een permanent bestand is, wordt dat bestand niet meer als een uitvoer maar als een interne logische file gezien. |
| - OPEN INPUT/ OUTPUT/ I-O/ EXTEND | - Een master file die geopend en gebruikt wordt voor lezen of schrijven telt men als een intern logisch bestand. |
| - CREATE VIEW | - Volgens de definitie van een bestand in FP analyse is elke logische gebruikersview een afzonderlijk intern logisch bestand. Dus tellen wij voor elke gecreëerde view een bestands functiepunt. |
| - CREATE TABLE | - Zie paragraaf 3.3 |
| - SET CONNECTION | - Is een SQL commando dat het mogelijk maakt om gegevens uit een andere database op te vragen. Verricht ongeveer hetzelfde werk als CALL in COBOL. Dus wordt er voor de zendende database een interface en voor de ontvangende een bestands functiepunt geteld. |
| - INCLUDE | - Met dit statement wordt er in een COBOL programma een database opgenomen. Die database wordt dan als een intern bestand beschouwd. Maar voor het opnemen van een database in een applicatie moet er eerst een SQL communicatie domein gereserveerd worden en dat gebeurt d.m.v. INCLUDE SQLCA statement. Deze INCLUDE wordt dan niet als een file geteld. |

| | | |
|---|---|---|
| - SET FILE OPEN | - | Zie paragraaf 3.2. |
| - CALL en - LINK | - | Zie paragraaf 3.2. |
| - XCTL | - | Zie paragraaf 3.2. |

## 3.6 Kolom "Interface"

| | | |
|---|---|---|
| - CALL | - | Als een programma aangeroepen wordt vanuit een master file dan rekent de applicatie waarin het aangeroepen programma zich bevindt een interface functiepunt. |
| - OPEN INPUT/ OUTPUT/ I-O/ EXTEND | - | Zie het OPEN statement in de vorige paragraaf. |
| - SET CONNECTION | - | Zie beschrijving in de vorige paragraaf. |
| - INCLUDE | - | Zie paragraaf 3.5. |
| - SET FILE OPEN | - | Zie paragraaf 3.2. |
| - CALL - LINK | - | Zie paragraaf 3.2. |
| - XCTL | - | Zie paragraaf 3.2. |
| - ALLOCATE | - | Maakt communicatie met een ander systeem mogelijk en daardoor wordt dit statement als een interface beschouwd. |
| - CONNECT PROCESS | - | Start het transactie proces met een extern systeem op en dat in dan een interface functiepunt waard. |

Nu we elk gekozen statement hebben toegelicht, moet nog opgemerkt worden dat er in COBOL code alleen ***verschillende*** functionele objecten geteld horen te worden. Bij voorbeeld als we twee DISPLAY blokken vinden die dezelfde boodschap naar het scherm sturen wordt dat als één uitvoer beschouwd.

## 3.7 Complexiteit van de functionele parameters

We hebben in Hoofdstuk 2 al gezien dat de functionele factoren nadat ze gevonden zijn, gesorteerd worden op complexiteit. Nou is het de vraag of de complexiteit bepaald kan worden voor de objecten die geëxtraheerd worden uit COBOL broncode.

De functiepuntenanalyse kent twee stappen in het bepalen van complexiteit. Ten eerste moet er voor elk functioneel object gekeken worden naar het aantal gebruikte data items en het aantal gerefereerde bestanden. Maar hoe krijgen we deze informatie uit de broncode?

De opdracht is redelijk eenvoudig voor de tabellen gecreëerd binnen een applicatie. Er wordt namelijk gekeken hoeveel kolommen (data items) een tabel heeft en omdat de betreffende datastructuren ontstaan en onderhouden worden binnen de applicatie hebben ze geen refererende bestanden. Dus hun complexiteit is of eenvoudig of gemiddeld.

Voor de databases opgenomen in een applicatie moet men naast het aantal kolommen ook het aantal bestanden weten die dezelfde database d.m.v. INCLUDE gebruiken.

Het ACCEPT statement is per definitie eenvoudig en de complexiteit van DISPLAY blokken wordt bepaald op basis van het aantal verschillende regels dat een blok bevat. De complexiteit van SELECT hangt van het aantal opgevraagde items en het aantal geraadpleegde tabellen af. Bij CALL kijken we naar het aantal programma's die hetzelfde programma aanroepen en het aantal verschillende parameters dat in de optie USING van CALL voorkomt.

Over de complexiteit van statements die bestandsnamen als parameters hebben kan weinig gezegd worden. Men zou in dat geval meer informatie over de betreffende files moeten hebben en dat soort gegevens kan of uit de documentatie of van de ontwikkelaars verkregen worden.

We kunnen elk functioneel object dat gevonden is maar niet gesorteerd op complexiteit tot *"gemiddeld"* verklaren of kijken hoe ingewikkeld het onderzochte systeem is en dan aan de factoren die nog gesorteerd moeten worden dezelfde complexiteit toekennen als die van de applicatie. Dan kunnen we het aantal functiepunten uitrekenen.

Om dat getal nauwkeuriger te maken moeten we de tweede stap van het bijstellen toepassen en dat is alleen mogelijk als we over juiste informatie beschikken. Men kan deze gegevens krijgen uit de documentatie waarin de ontwikkelingseisen zijn beschreven en/of uit de gedetailleerde beschrijving van de applicatie. Meestal worden deze documenten niet verstrekt door de eigenaar van software. En als de beschrijvingen wel bestaan wat vaak niet het geval is, zijn ze dan verouderd.

In geval er gebrek aan documentatie is, kan men alleen het domein bepalen waarin het uiteindelijke resultaat zal liggen. Maar volgens J.B. Dreger [1] is het veel belangrijker om de functionele parameters te vinden dan het sorteren daarvan. Dus als we met onze methode erin slagen om deze objecten te vinden, hebben we ook een goed resultaat bereikt.

# Hoofdstuk 4

# Resultaten

Voor het valideren van de meettechniek gepresenteerd in het vorige hoofdstuk is een applicatie gebruikt waarvoor het aantal functiepunten al eens bepaald is. De precieze manier van tellen is ons niet bekend, maar we weten wel dat de applicatie (verder in de tekst ook NW-applicatie genoemd) of een deel daarvan 279 functiepunten telt en gaan ervan uit dat dit getal zeer nauwkeurig is bepaald. Verder weten we dat alle drieëntwintig geleverde COBOL programmabestanden en negentien copybooks tot dezelfde applicatie behoren maar of er in de NW-applicatie nog meer programma's zitten is ons niet bekend. Van de gekregen programmafiles zijn er vijftien COBOL bestanden gegenereerd uit Telon – ook wel NW-listings genoemd - en de rest zijn gewone COBOL sources. Hierbij moet ook gezegd worden dat de COBOL bestanden gegenereerd uit Telon voor meer dan 50% uit code bestaan die geen reguliere COBOL is maar een soort codeadministratie die de compiler voor eigen gebruik aanmaakt. Voor de metingen is daarom alleen naar de reguliere COBOL code gekeken.

Om de beschrijving van het verrichte werk makkelijker te kunnen volgen worden hieronder alle bestandsnamen van de NW-applicatie genoemd.

- COBOL sources zijn: NW200, NW201, NW202, NW210, NW212, NW213, NW290 en NW301
- NW-listings zijn: NW100A, NW100B, NW100C, NW100D, NW100E, NW100F, NW100G, NW100I, NW100J, NW100K, NW100L, NW100M, NW100N, NW100O, NW100P
- Copybooks zijn: A0021296, A0021861, A0021862, A0030761, A0030770, A0030774, A0030777, A0030778, A0030781.CPY, A0030783, A0030784, A0048709, C01899C, GUTA00, GUTB00, NW201E, NW201S, NW210E, NW210S

De tellingen zijn op drie manieren uitgevoerd. In de eerste telling zijn met behulp van het *grep* commando de nodige COBOL en SQL statements uit de broncode gefilterd en de tweede telling is gedaan m.b.v. COBOL documentatiegenerator – DocGen. De derde manier van tellen is gebaseerd op de backfiring methode.

## 4.1 Handmatige tellingen

Met het grep commando hebben we in eerste instantie het volgende gekregen:
- 37 SEARCH statements naar verschillende objecten.
- 33 CALL statements naar verschillende programma's.
- 56 meldingen m.b.v. DISPLAY statement.
- 1 INCLUDE statement voor de data base VNW200GTBOEKDAT.
- 3 SELECT statements voor het ophalen van gegevens uit VNW200GTBOEKDAT.

Na wat meer analyse is gebleken dat:
- Alle 37 SEARCH statements refereren naar tabellen die ontstaan zijn met het REDEFINES statement van COBOL. Zulke datastructuren worden volgens de regels **niet** meegeteld .
- 18 van de 33 CALLs hebben parameters die niet in de 23 NW-bestanden voorkomen en dus als externe programma's beschouwd moeten worden. De andere 15 CALLs roepen programma's aan die binnen dezelfde applicatie zitten[4]. We hebben al eerder gezegd dat er voor het tellen van functiepunten alleen bestanden van buiten de applicatie van belang zijn en dat houdt in dat er maar 18 CALL statements in de telling beschouwd worden.
- In de genoemde 56 schermuitvoeren zijn sommige dubbel geteld. Na wat vergelijkingen bleven er uiteindelijk 30 ***verschillende*** schermuitvoeren over.

Nou hebben we dus:
- 18 CALL statements van verschillende externe programma's.
- 30 verschillende schermuitvoeren.
- 1 INCLUDE statement voor het includeren van de data base VNW200GTBOEKDAT.
- 3 SELECT statements voor het ophalen van gegevens uit VNW200GTBOEKDAT.

Met de verkregen applicatie is geen enkele documentatie tot onze beschikking gesteld en daarom moesten we zelf uitzoeken hoe we met de programmabestanden hoorden om te gaan. Dat resulteerde in het maken van twee berekeningen die hieronder beschreven zullen worden.

Het gebrek aan documentatie heeft het verder ook onmogelijk gemaakt om het gekregen aantal functiepunten m.b.v. 14 complexiteitsfactoren bij te stellen. Zo verkrijgen we dus alleen het bereik waarin het uiteindelijke aantal functiepunten zich bevindt.

Omdat de genoemde berekeningen gebaseerd zijn op verschillende benaderingen geven we eerst (in de Tabel 8) een overzicht van de uitgangspunten voor zowel de eerste als voor de tweede berekening.

[4] Het programma NW100H is niet aan ons ter beschikking gesteld maar volgens de interne regels van de eigenaar van de software geven de eerste twee letters van de naam van een programma de applicatie aan. Dus hebben we geconcludeerd dat het programma NW100H ook een intern programma is.

| BEREKENING 1 | BEREKENING 2 |
|---|---|
| Alle 18 CALLs zijn als interfaces voor NW-applicatie geteld d.w.z. alle programmabestanden worden als master files beschouwd. | 15 COBOL listings worden transactie files en CALLs van deze files zijn geen interfaces maar invoer zoals het in paragraaf 2.3 staat. Omdat het om COBOL files gaat wordt er vanuit gegaan dat er geen dataconversie plaatsvindt en zijn er dus geen extra interfaces geteld.<br>De CALLs in source files blijven interfaces. |
| Alle 30 schermuitvoeren met DISPLAY zijn zoals in Tabel 7a staat als uitvoer beschouwd. | Schermuitvoeren met DISPLAY zijn ook als uitvoer beschouwd. |
| Voor INCLUDE is er een file en een interface berekend zoals in Tabel 7b is aangegeven. | Voor INCLUDE is ook een file en een interface berekend. |
| Voor 3 SELECTs zijn drie navragen berekend omdat deze statements verschillende gegevens uit dezelfde database opvragen. | Voor SELECTs zijn er ook drie navragen geteld. |
| | 8 COBOL bronfiles worden beschouwd als interne logische bestanden. |

Tabel 8: uitgangspunten voor handmatige tellingen

Tevens zijn beide berekeningen grafisch gepresenteerd in Figuur 1 en Figuur 2 aan het eind van deze paragraaf.

### 4.1.1 Berekening 1

De gegevens die we nu hebben gekregen zijn:
Geen invoer, 30 uitvoeren, 3 navragen, 1 bestand en 19 interfaces.

Er moet nou natuurlijk naar de complexiteit van de gevonden factoren gekeken worden.

Invoer (IT)

De applicatie heeft geen invoer dus valt er niets voor de complexiteit uit te rekenen.

Uitvoer (OT)

Elke uitvoer gebruikt tussen 0 en 7 items en geen bestanden. Dus volgens Tabel 3 hebben alle dertig uitvoeren simpele complexiteit.

Navragen (QT)

Elk invoergedeelte van de navragen gebruikt 3 data items en refereert naar een bestand (DB zelf) en elk uitvoer produceert een data item. Alle drie invoergedeeltes zijn verschillend terwijl elke SELECT hetzelfde formaat en proceslogica van de uitvoer heeft. Maar vanwege de invoergedeelte moeten we drie navragen tellen die dan simpele complexiteit hebben.

<u>Bestanden (FT)</u>

De database VNW200GTBOEKDAT heeft één logisch record bestaand uit 3 items. Dus hebben we hier te maken met een simpel bestand.

<u>Interfaces (EI)</u>

Het interface dat aan de database toegekend wordt is simpel. Verder hebben we 6 CALLs die door maar een programmafile gedaan worden en daarbij wordt er tussen 1 en 8 data items gebruikt. Dus alle 6 interfaces zijn simpel. Dan hebben we twee interfaces die in de relatie met twee bestanden staan en elk gebruikt 4 data items. De complexiteit van deze interfaces is weer simpel. Er is één CALL die naar 3 bestanden refereert en daarbij worden 4 data items gebruikt. Deze interface wordt volgens Tabel 6 ook als een simpele interface beschouwd. Vier CALLs staan in relatie met 13 files en ze maken gebruik van 3,4 of 6 parameters dus hebben ze gemiddelde complexiteit. De programma's die door 14 programmabestanden aangeroepen worden en minder dan 8 parameters gebruiken worden ook als gemiddelde interfaces beschouwd. De programma's UT100 en CBLTDLI[5] die door 17 respectievelijk 18 verschillende applicatieprogramma's aangeroepen worden en daarbij meer dan 20 data records gebruiken zijn complexe interfaces.

Uiteindelijk krijgen we het volgende.

| aantal | Complexiteit | functionele factor | weegfactor | Aantal FP |
|---|---|---|---|---|
| 0 | n.v.t. | IT | n.v.t. | 0 |
| 30 | Simpel | OT | 4 | 120 |
| 3 | Simpel | QT | 4 | 12 |
| 1 | Simpel | FT | 7 | 7 |
| 10 | Simpel | EI | 5 | 50 |
| 7 | Gemiddeld | EI | 7 | 49 |
| 2 | Complex | EI | 10 | 20 |
| | | | **TOTAAL** | **258** |

Uitgerekend met 14 complexiteitsfactoren zou het bovenste getal in een bereik tussen **168** en **348** komen.

Het verkregen getal (258) zit 7,53% naast het resultaat van de software-eigenaar of m.a.w. het verschilt een factor 1,08 van 279.

Maar als we het percentage van de in de NW-applicatie voorkomende functionele objecten gaan vergelijken met de statistieken van Dreger [1] (Tabel 10) krijgen we het volgende beeld:

| Functionele factor | Resultaat van de eerste berekening in % |
|---|---|
| Invoer (IT) | 0 |
| Uitvoer (OT) | 46,51 |
| Navraag (QT) | 4,65 |
| Bestand (FT) | 2,71 |
| Interface (EI) | 46,12 |

Tabel 9: vertegenwoordiging van de functionele objecten volgens berekening 1

[5] Het programma NW210 roept *"by reference"* HCBLTDLI terwijl alle andere programma's *"by value"* 'CBLTDLI' aanroepen. Maar er wordt in feite hetzelfde programma aangeroepen.

| Functionele factor | Gewogen gemiddelde in % | Domein in % |
|---|---|---|
| Invoer (IT) | 31 | 18-49 |
| Uitvoer (OT) | 24 | 13-55 |
| Navraag (QT) | 8 | 0-22 |
| Bestand (FT) | 36 | 7-52 |
| Interface (EI) | 1 | 0-20 |

Tabel 10: vertegenwoordiging van functionele factoren in applicaties

Het lijkt erop dat het aantal invoeren, bestanden en interfaces niet helemaal klopt met de getallen die uitgerekend zijn. Helaas is het aantal objecten in de applicatie ons niet bekend. Dus het is ook niet uitgesloten dat dit resultaat goed is gezien het feit dat we misschien over een deel van de gehele applicatie beschikken.

Toch is de broncode nogmaals goed bekeken en zijn we vanuit een andere invalshoek tot de volgende berekening gekomen.

### 4.1.2 Berekening 2

De gegevens waar we nou over beschikken zijn:
15 invoeren, 30 uitvoeren, 3 navragen, 9 bestanden en 6 interfaces.

Nou gaan we voor de opnieuw gedefinieerde functionele objecten de complexiteit bepalen.

Invoer (IT)

De applicatie heeft nu 15 invoeren waarvan de 4 met minder dan drie bestanden in de relatie staan en 3 of 4 data records gebruiken. Één invoer refereert aan 1 bestand en gebruikt 8 data items. Deze 5 invoeren worden dan als simpel beschouwd. Verder zijn er 6 gemiddelde invoeren die naar hoogstens 14 bestanden refereren en tussen 0 en 4 items gebruiken. En als laatste hebben we nog 4 gevallen waarin naar 10, 13, 14 en 15 bestanden gerefereerd wordt en respectievelijk 4, 6, 7 en 4 parameters gebruikt worden. Die invoeren worden dan complex genoemd.

Uitvoer (OT) en navragen (QT)

Uitvoer en navragen zijn dezelfde als in berekening 1.

Bestanden (FT)

De database VNW200GTBOEKDAT heeft één logisch record bestaand uit 3 items. Dus hebben we hier te maken met een simpel bestand. Voor de 8 COBOL bestanden kon niet uit de broncode geconcludeerd worden wat voor complexiteit ze hebben. Maar sinds het in het algemeen om een simpele applicatie gaat (geconcludeerd op basis van het bekende aantal functiepunten) en de programma's redelijk klein zijn, worden ze als simpele bestanden beschouwd.

Interfaces (EI)

Het interface dat aan de database toegekend wordt is simpel. Drie andere programma's worden door maar een programma aangeroepen en daarbij gebruiken ze minder dan 5 data

items. Dus hebben we hier met drie simpele interfaces te maken. Het programma CBLTDLI[6] staat in relatie tot 3 programmabestanden en gebruikt 41 data records dus heeft dit interface een gemiddelde complexiteit. Ten slotte het programma UT100 dat door 7 NW-files aangeroepen wordt en daarbij meer dan 20 data items gebruikt, wordt als een complex interface beschouwd.
Dan krijgen we het volgende.

| aantal | complexiteit | functionele factor | weegfactor | Aantal FP |
|---|---|---|---|---|
| 5 | simpel | IT | 3 | 15 |
| 6 | gemiddeld | IT | 4 | 24 |
| 4 | complex | IT | 6 | 24 |
| 30 | simpel | OT | 4 | 120 |
| 3 | simpel | QT | 4 | 12 |
| 9 | simpel | FT | 7 | 63 |
| 4 | simpel | EI | 5 | 20 |
| 1 | gemiddeld | EI | 7 | 7 |
| 1 | complex | EI | 10 | 10 |
| | | | **TOTAAL** | **295** |

Uitgerekend met 14 complexiteitsfactoren zou het bovenste getal in een bereik tussen **192** en **398** komen.

Het verkregen aantal functiepunten (295) zit 5,73% naast het resultaat van de software-eigenaar of m.a.w. het verschilt een factor 0,94 van 279.

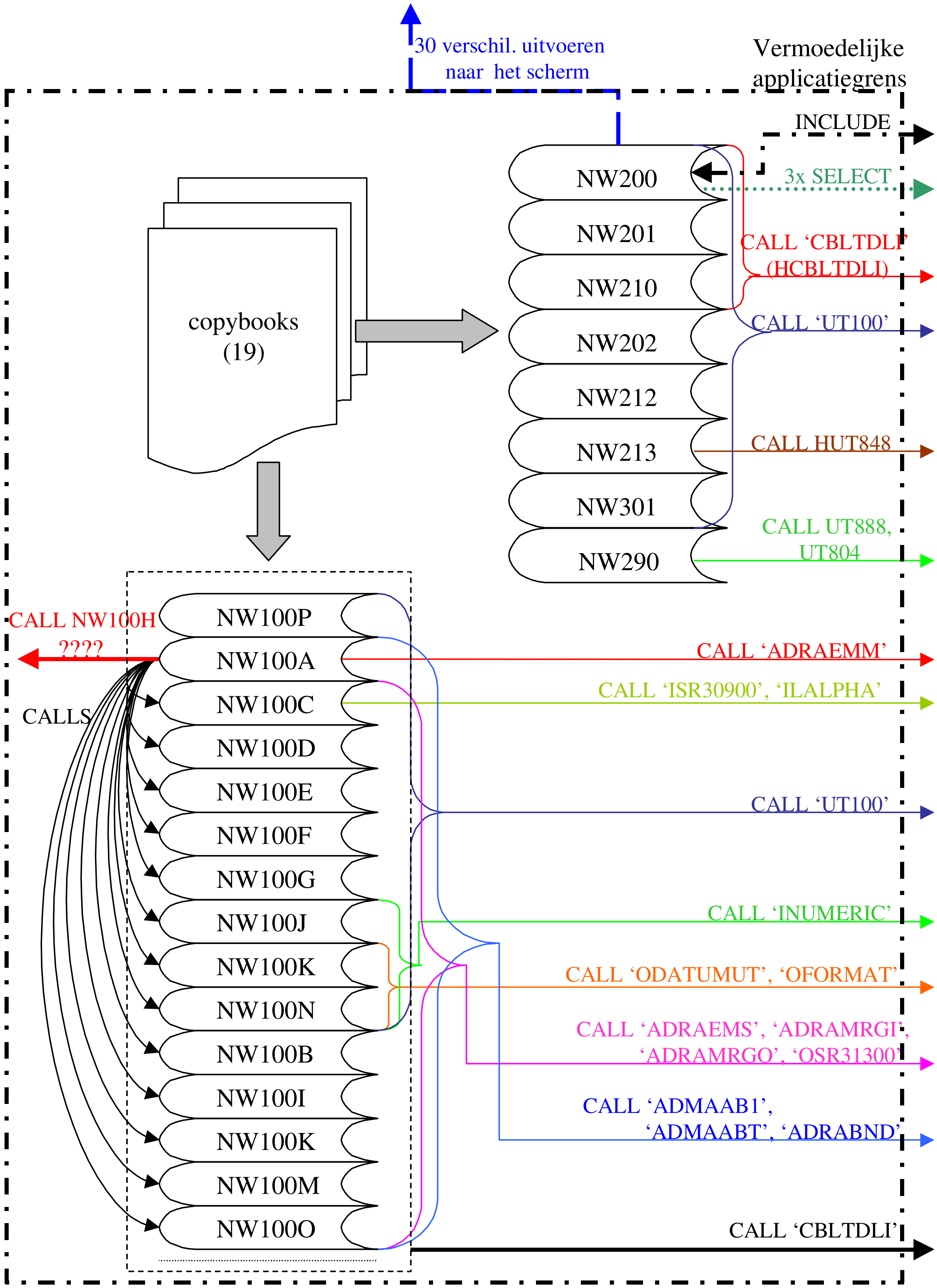
Als we deze keer weer het percentage van FP objecten uitrekenen krijgen we:

| Functionele factor | Resultaat van de tweede berekening in % |
|---|---|
| Invoer (IT) | 21,62 |
| Uitvoer (OT) | 40,54 |
| Navraag (QT) | 4,05 |
| Bestand (FT) | 21,28 |
| Interface (EI) | 12,50 |

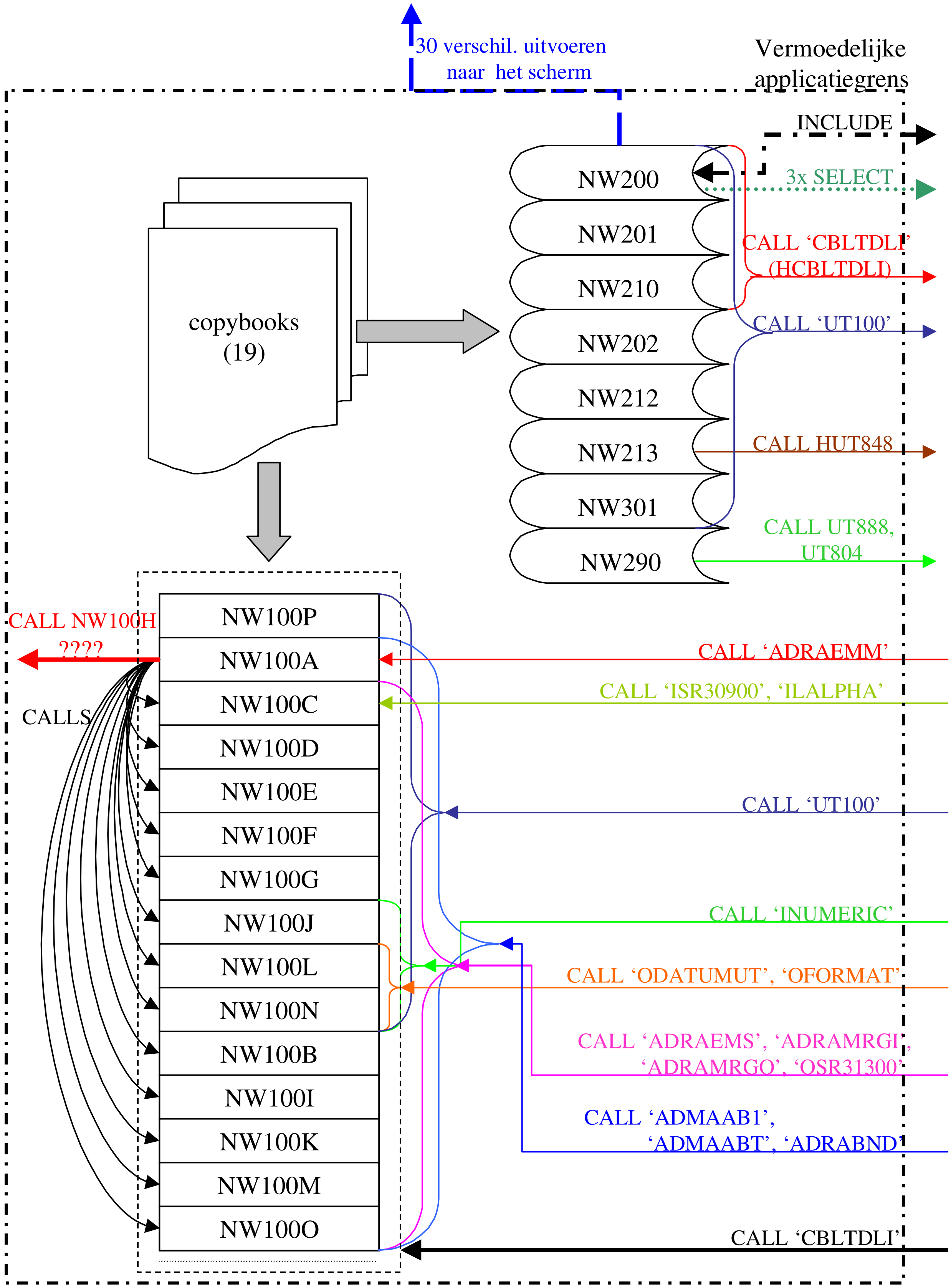
Tabel 11: vertegenwoordiging van de functionele objecten volgens berekening 2

Zo te zien zitten deze resultaten wat dichterbij de statistieken van Dreger (Tabel 10). Maar om erachter te komen welke van deze twee berekeningen op de NW applicatie van toepassing is, hebben we meer informatie over de applicatie nodig.

[6] zie vorige voetnoot.




Figuur 1: berekening 1



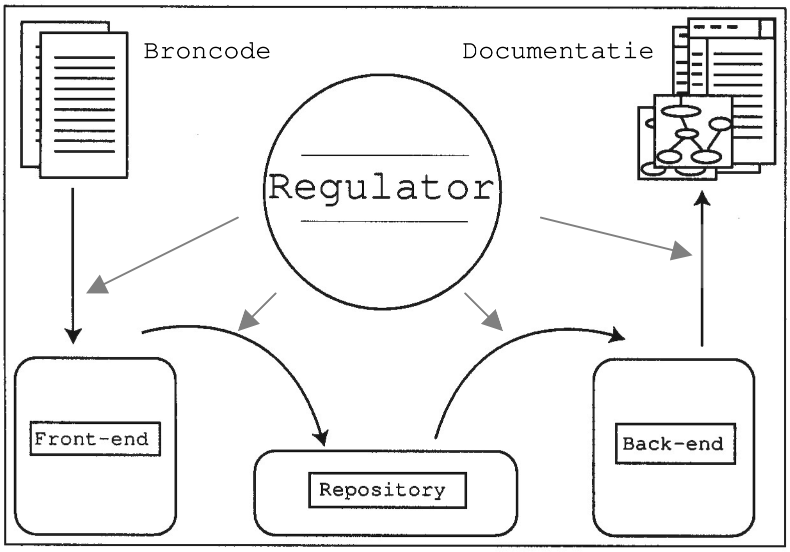

Figuur 2: de tweede berekening

## 4.2 Tellingen met de DocGen technologie

Voordat we de resultaten introduceren die met de DocGen technologie zijn verkregen, willen we een beschrijving van dat tool geven.

### 4.2.1 DocGen

DocGen genereert automatisch technische documentatie uit broncode. Zo krijgen we informatie over het systeem in z'n geheel b.v. programma afhankelijkheden en statistieken maar ook informatie over de programma's afzonderlijk. Alle resultaten verkregen met dat tool worden zowel tekstueel als grafisch in een web browser gepresenteerd. Om al dat werk te verrichten heeft DocGen vier onderdelen (Figuur 3 [14]) die allemaal hun eigen taak hebben in het bereiken van één doel – het genereren van technische documentatie.




Figuur 3: onderdelen van DocGen

Het eerste subsysteem van DocGen – *front-end* – moet ervoor zorgen dat gegevens uit de broncode geëxtraheerd worden. *Repository* slaat de uitvoer van *front-end* op in een MySql relationele database en deze gegevens worden later door *back-end* gebruikt om m.b.v. HTML een gebruikersinterface te creëren. Tijdens het hele proces zorgt *regulator* ervoor dat o.a. de broncode ingelezen wordt, relationele database gevuld wordt en de resultaten in een web browser gepresenteerd worden.

### 4.2.2 Van DocGen naar FP-Gen

Hoewel DocGen in staat is om veel informatie uit broncode te halen, was het niet mogelijk om met deze documentatie generator functiepunten te tellen. Daarom hebben we m.b.v. DocGen een prototype gemaakt om de nodige statements uit de NW-applicatie te extraheren en op basis van de gekregen gegevens functiepuntentelling te genereren. Dat tool-prototype hebben we FP-Gen genoemd.

FP-Gen heeft in de NW-applicatie gevonden:

- 19 CALLs[7]
- 24 DISPLAY blokken
- 1 INCLUDE
- 3 SELECTs
- 8 master files en
- 37 SEARCHes

De tabellen die de parameters van het SEARCH statement zijn, worden herdefinieerd en als zulke worden ze in de functiepuntenanalyse niet geteld. Programma's aangeroepen vanuit NW-listings worden als invoer beschouwd en programma's aangeroepen vanuit NW-sources zijn interfaces. FP-Gen vond verder 24 verschillende schermuitvoeren en 3 navragen m.b.v. het SELECT statement. In de applicatie is maar één database opgenomen waarvoor we een interface en een bestands functiepunt moeten tellen. En ten slotte hebben we hier nog 8 COBOL programmabestanden die volgens de FP methode interne logische files zijn.

De complexiteit van alle functionele parameters kon met FP-Gen semi-automatisch bepaald worden. Zo kregen we:

<u>Invoer (IT)</u>

Van de 15 invoeren telde FP-Gen automatisch 5 simpele, 5 gemiddelde en 5 moeilijke.

<u>Uitvoer (OT)</u>

Voor de gevonden DISPLAY blokken kon de complexiteit niet automatisch afgeleid worden. Maar omdat de NW-aaplicatie klein een redelijk simpel is, wordt de schermuitvoer ook als eenvoudig beschouwd.

<u>Navraag (QT)</u>

De tellingen van navragen zijn met FP-Gen dezelfde als de handmatige. We hebben dus drie simpele navragen verkregen met het SELECT statement.

<u>Bestanden (FT)</u>

FP-Gen beschouwt de in de applicatie opgenomen database (VNW200GTBOEKDAT ) als een simpele datastructuur en dan wordt deze ook als eenvoudige bestand gezien. Over de complexiteit van de master files kon ons tool niets zeggen dus hebben we geredeneerd op dezelfde manier als voor de uitvoeren. D.w.z. dat de interne logische bestanden simpele complexiteit toegewezen krijgen.

<u>Interfaces (EI)</u>

De complexiteit van de VNW200GTBOEKDAT database als een interface is analoog aan de complexiteit van dezelfde datastructuur als een bestand. Van de andere door FP-Gen gevonden interfaces telde hij 4 simpele en 2 complexe.

[7] programma's CBLTDLI en HCBLTDLI zijn als twee verschillende programma's gezien door FP-Gen .

De tweede stap die het functiepunten generatie tool-prototype uitvoert is de vermenigvuldiging van alle getelde functionele parameters met de bijbehorende weegfactoren. En dan krijgen we:

| Aantal | complexiteit | functionele factor | weegfactor | Aantal FP |
|---|---|---|---|---|
| 5 | simpel | IT | 3 | 15 |
| 5 | gemiddeld | IT | 4 | 20 |
| 5 | complex | IT | 6 | 30 |
| 24 | simpel | OT | 4 | 96 |
| 3 | simpel | QT | 4 | 12 |
| 9 | simpel | FT | 7 | 63 |
| 5 | simpel | EI | 5 | 25 |
| 2 | complex | EI | 10 | 20 |
| | | | **TOTAAL** | **281** |

FP-Gen kon de tweede stap van de berekening (vaststellen van 14 complexiteits-factoren) niet automatisch uitvoeren omdat de documentatie die hij genereert onvoldoende is voor het beoordelen van de complexiteit van een applicatie in zijn geheel. Het was in ieder geval mogelijk om het domein uit te rekenen waarin het uiteindelijke resultaat zich bevindt en dat bereik is **183 – 379**.

Wat opvalt is dat onze automatische telling maar 2 functiepunten verschilt van de telling van de software-eigenaar. Uitgedrukt in procenten is dat 0,72% of een factor 0,99.

Verder zijn we ook geïnteresseerd in hoeverre mate de vertegenwoordiging van door FP-Gen getelde functionele elementen overeenkomt met de statistieken uit Tabel 10.

| Functionele factor | Resultaat van FP-Gen berekening in % |
|---|---|
| Invoer (IT) | 23,13 |
| Uitvoer (OT) | 34,16 |
| Navraag (QT) | 4,27 |
| Bestand (FT) | 22,42 |
| Interface (EI) | 16,01 |

Tabel 12: vertegenwoordiging van de functionele objecten volgens de FP-Gen telling

De verkregen resultaten lijken sterk op die van berekening 2 en kloppen ook volgens de statistieken.

## 4.3 Meten met backfiring

Nadat de tellingen met de ontwikkelde meetmethode zijn uitgevoerd, willen we kijken welke uitkomst de backfiring methode zal opleveren. Voor het toepassen van die metriek moeten we het totale aantal logische regels code in de NW-applicatie weten even als het aantal regels die men in een COBOL programma moet schrijven om één functiepunt te krijgen. Voor het tweede getal is er een tabel geraadpleegd uit het boek *Applied Software Measurement* van C. Jones [2]. Daar hebben we gevonden dat er gemiddeld 107 regels

COBOL code nodig zijn voor één functiepunt. Verder zijn de COBOL telregels van het bedrijf *Software Productivity Research* gebruikt [2] en uiteindelijk zijn er in de NW-applicatie 36086 fysieke regels (COBOL) code geteld. Hierbij zijn lege en commentaar regels niet meegerekend. Als we dat laatste getal delen door 107 krijgen we **337** en dat is het aantal functiepunten gemeten met de backfiring methode.

Vergelijking van dat resultaat met 279 functiepunten die door de software-eigenaar zijn gemeten, levert het verschil van 20,79% ofwel factor 0,83 op.

## 4.4 Vergelijking uitkomsten

| | Gemeten door de software-eigenaar | Berekening 1 | Berekening 2 | DocGen telling | Tellen met backfiring |
|---|---|---|---|---|---|
| Totaal aantal FP | 279 | 258 | 295 | 281 | 337 |
| Bereik | Onbekend | 168-348 | 192-398 | 183-379 | 212-555 |
| Meetfout in % | 0 | 7,53 | 5,73 | 0,72 | 20,79 |
| Factor | 1 | 1,08 | 0,94 | 0,99 | 0,83 |

Tabel 13: meetresultaten

Als we nou alle verkregen resultaten vergelijken (Tabel 13) met het reeds bekende aantal functiepunten zien we dat alle uikomsten die de ontwikkelde methode oplevert in de buurt van 279 zitten onafhankelijk van de manier waarop men aan het aantal functiepunten kwam. Hoewel de handmatige metingen zeer goede uitkomsten hebben gegeven, is de FP-Gen telling diegene die de kleinste meetfout heeft (0,72%) en daarmee ook het resultaat behaalt dat het meest dichtbij het getal gegeven door de software-eigenaar komt.

Aan de andere kant levert de backfiring methode een redelijk grote meetfout op (21%). De reden daarvoor kan zijn dat het aantal regels code niet goed is bepaald maar we hebben dat probleem van backfiring al eerder genoemd en gezegd dat deze metriek niet nauwkeurig genoeg is voor serieuze functiepuntenstudies wegens zijn afhankelijkheid van het aantal regels code. Maar toch kunnen we deze metriek wel gebruiken voor grootschalige metingen, educatie en de studies waarin het niet zozeer om de nauwkeurigheid maar wel om de snelheid gaat.

En als het en snel en precies moet, zien we dat er gelukkig een andere mogelijkheid bestaat om de omvang van software te meten.

# Hoofdstuk 5

# Conclusie

Het doel van het verrichte onderzoek was het vinden van een manier waarop functiepunten automatisch geteld kunnen worden op basis van bestaande broncode. Met behulp van het *grep* commando en het prototype van een functiepunten generator FP-Gen zijn uit een COBOL applicatie de statements die in het systeem geïmplementeerde businessfuncties aanduiden, geëxtraheerd. Daarna is het aantal functiepunten per functionele parameter uitgerekend en ook het totale aantal functiepunten die de applicatie bevat.

Voor het systeem waarop de ontwikkelde methode is toegepast, was het aantal functiepunten al eerder berekend. Er zijn drie verschillende tellingen uitgevoerd met de nieuwe methode en die leverden allemaal verschillende resultaten op. Alle verkregen uitkomsten zijn vergeleken met het door de software-eigenaar geleverde getal. Het bleek dat de getallen die uit onze methode resulteerden heel dicht in de buurt kwamen van de uitkomst die ons door de software-eigenaar bekend is gemaakt. Sterker nog, als we de ontwikkelde techniek vergelijken met backfiring die tot nu toe de enige manier was om functiepunten op basis van bestaande sources 'automatisch' te tellen, zien we dat onze resultaten veel nauwkeuriger zijn. Bovendien kan men de functionele parameters niet vinden met backfiring maar wel met de uit het onderzoek resulterende methode. Het enige voordeel van de backfiring metriek is dat hij taalonafhankelijk is terwijl onze methode alleen op COBOL, SQL en CICS systemen toegepast kan worden. Maar nu weten we welk soort statements van belang is voor de functiepuntentechniek. Dus is het implementeren van de ontwikkelde methode voor andere programmeertalen geen probleem meer.

Een ander obstakel om deze nieuwe methode wereldwijd in gebruik te stellen is het bestaan van meerdere non-profit verenigingen voor het tellen van functiepunten die allemaal verschillende telregels hanteren. Ons onderzoek is gebaseerd op de regels van de IFPUG en dat betekent dat een andere organisatie de ontwikkelde methode niet kan gebruiken.

De verrichte berekeningen zouden waarschijnlijk nog preciezer zijn als we in het onderzoek over meer informatie betreffende de te meten applicatie hadden kunnen beschikken. Het is namelijk niet zo eenvoudig om uit broncode alleen te concluderen hoe ingewikkeld een applicatie is (er moeten extra tools gebruikt worden zoals b.v. tool voor het berekenen van McCabe complexiteit) en welk soort werk de betreffende applicatie of de delen daarvan verrichten en deze factoren zijn ook van belang voor het tellen van functiepunten. Dat betekent verder dat er in een aantal gevallen subjectieve beslissingen over de complexiteit van sommige componenten (b.v. master files) genomen moeten worden. Het gevolg daarvan is dat het verkrijgen van een exact aantal functiepunten gebaseerd is op een telproces dat niet volledig geautomatiseerd kan worden indien men niet over de juiste documentatie beschikt. Toch is het wel mogelijk om het bruto aantal functiepunten automatisch uit te rekenen even als het bereik waarin het uiteindelijke resultaat zich bevindt. En dat is nog steeds beter dan de uitkomsten die de backfiring methode geeft en sneller dan de handmatige telling.

De onderzoeksresultaten zouden nog gevalideerd moeten worden door het toepassen van de ontwikkelde methode op nog minstens één businessapplicatie. Maar helaas is het ons niet gelukt om aan meer broncode te komen waarvoor men het aantal functiepunten al heeft uitgerekend. Die tak van software engineering is vrij jong in Nederland en er zijn weinig bedrijven die dat soort werk proberen te doen. De oorzaak daarvoor zou kunnen liggen in het feit dat tot nu toe de enige mogelijke – handmatige - manier van precies tellen te veel tijd en geld kostte.

## Toekomstig werk

Een geautomatiseerde methode voor het tellen van functiepunten zal van groot belang zijn voor het bedrijfsleven. De kosten voor de begroting van businessapplicaties zouden omlaag gaan terwijl de kwaliteit van metingen verbeterd wordt. Verder zou men in staat zijn budgetplannen, projectduur, onderhoudskosten enz. met grote nauwkeurigheid vast kunnen stellen.

Hoewel nog maar op één applicatie toegepast, legt de ontwikkelde methode een goede basis voor een tool dat in staat zou zijn onafhankelijk van de programmeertaal het aantal functiepunten in bestaande broncode te bepalen.

# Referenties

[1] Function Point Analysis - J. Brian Dreger - Prentice Hall-1989

[2] Applied Software Measurement - Capers Jones - McGraw-Hill – 1996

[3] COBOL - W.B.C. Ebbinkhuijsen - Samsom – 1979

[4] CICS Command Level Programming - Alida Jatich - John Wiley & Sons – 1991

[5] Information Systems Through COBOL - Andreas S. Philippakis en Leonard J. Kazimier - McGraw-Hill – 1978

[6] On-line Systems Design & Implementation Using COBOL and Command Level CICS - Charles J. Kacmar - Prentice Hall-1984

[7] DB2 for COBOL Programmer- part 1 - Curtis Garvin en Steve Eckols - Mike Murach & Associates – 1998

[8] DB2 for COBOL Programmer- part 2 - Curtis Garvin en Anne Prince - Mike Murach & Associates – 1998

[9] Structured COBOL Report Writer - David Schechter en George Yvkoff - Prentice Hall-1982

[10] Function Point FAQ - http://ourworld.compuserve.com/homepages/softcomp/

[11] What Are Function Points? - http://www.spr.com

[12] Fundamentals of Function Point Analysis by David Longstreet - http://www.ifpug.com/fpafund.htm

[13] SIG DocGen – http://www.software-improvers.com/docgen.html

[14] COBOL island grammars in SDF – Ernst-Jan Verhoeven – 31 augustus 2000

[15] Software Assessments, Benchmarks and Best Practices – Capers Jones – Addison Wesley - 2000