Bepaling van de geschiktheid van Oracle Forms applicaties voor inbeheername door middel van automatische code review volgens het SIG Maintainability model

Aart van den Dolder

30 augustus 2011

Tweejarige Master Software Engineering (deeltijd)

Afstudeerdocenten: Jurgen Vinju en Hans Dekkers

Begeleiders: Ingrid Jansen en Sybrand Houtsma

Opdrachtgever: Ordina Oracle Solutions, team Applicatie Support

Publicatiestatus: definitief

Universiteit van Amsterdam

Inhoud

Samenvatting

Onderzocht is de bruikbaarheid van automatische code-reviews voor het beoordelen van de risico's bij het in beheer nemen van klassieke Oracle-Forms applicaties. Hiervoor zijn 9 kleine tot middelgrote applicaties onderzocht, waarvan het beheer door dezelfde organisatie is uitgevoerd.

De automatische code-reviews zijn uitgevoerd volgens het SIG-maintainability model. Van elke applicatie zijn de scores bepaald in de twee dimensies van het model, namelijk:
- de statische code-eigenschappen: volume, complexiteit, duplicaten en unitgrootte.
- de ISO9126 subkarakteristieken: analyseerbaarheid, wijzigbaarheid en testbaarheid.
De bepaalde ISO9126 subkarakteristieken van de applicaties bleken weinig van elkaar te verschillen, terwijl er wel degelijk verschillen waren in de gemeten statische code-eigenschappen.

De statische code-eigenschappen die zijn gebruikt voor de bepaling van de ISO9126 subkarakteristieken zijn vervolgens vergeleken met gegevens voor de beheersinspanning die waren vastgelegd in een issuemanagementsysteem. Voor de meeste eigenschappen is hierbij geen significante correlatie gevonden met de totale onderhoudsinspanning. Voor twee eigenschappen, het totale aantal regels code en het percentage regels code in units groter dan 50 regels, zijn wel significante correlaties gevonden, maar hierbij bleken er ook applicaties te zijn waarvan het aantal bestede uren buiten het algemene patroon viel.

Uit de resultaten van dit onderzoek kan geconcludeerd worden dat voor de beoordeling van Oracle Forms applicaties het SIG-maintainability model niet nauwkeurig genoeg is. Ook de afzonderlijke statische code-eigenschappen zijn niet of in beperkte mate geschikt om de benodigde onderhoudsinspanning voor een applicatie te voorspellen. Waar wel een correlatie is gevonden met de onderhoudsinspanning, moet altijd rekening worden gehouden worden met de mogelijkheid dat een andere factor meer bepalend is voor de te verwachten onderhoudsinspanning.
Het SIG-maintainability model is dus om de benodigde onderhoudsinspanning voor een bepaalde applicatie te kunnen voorspellen beperkt bruikbaar. Mogelijk is het niet toereikend voor een heterogene taal als PL/SQL. De omvang en de complexiteit van de SQL-component van deze taal maken geen onderdeel uit van het model, terwijl het aannemelijk is dat deze factoren invloed hebben op de benodigde onderhoudsinspanning. Ook het gebruik van codegeneratie tijdens het ontwikkelen van een applicatie kan ervoor zorgen dat eigenschappen als de cyclomatische complexiteit en het percentage duplicaten geen indicatie vormen voor de onderhoudbaarheid van een applicatie.

Naast onderhoud is in dit onderzoek ook gekeken naar een ander aspect van beheer, namelijk de afhandeling van incidenten. Er is geen verband gevonden tussen de beheersinspanning m.b.t. incidenten en de gemeten code-eigenschappen. Incidenten zijn dus niet een gevolg van gebrekkige codekwaliteit.

Voorwoord

Dit onderzoek heeft plaatsgevonden in een voor mijn gezin zeer intensieve periode, waarin ik regelmatig mijn werk opzij moest zetten om mijn gezin tot steun te kunnen zijn. Ik ben dan ook mijn werkgever Ordina, in de persoon van mijn toenmalige Business Unit manager Marco Rutters, zeer erkentelijk dat ik hiervoor de ruimte heb gekregen. Dat geldt meer in het algemeen ook voor de gelegenheid die mijn werkgever mij heeft geboden om deze master-opleiding te kunnen doen.

Het was bijzonder dat dit onderzoek in Groningen plaatsvond, dezelfde stad waar mij ruim 20 jaar geleden na een succesvolle HTS-stage aan de faculteit Technische Scheikunde van de RUG op het hart werd gedrukt om mijn academische opleiding te voltooien (1). Dat dit op een heel ander vakgebied zou zijn kon ik toen niet vermoeden. De kennis van de chemometrie die ik destijds heb opgedaan heb ik bij dit onderzoek dankbaar kunnen gebruiken.

Een groot deel van de tijd die ik in dit onderzoek heb gestoken is helaas niet zichtbaar in dit verslag. Deze tijd is gaan zitten in de bouw van de analysetool, zoals die beschreven is in hoofdstuk 4. En passant heb ik hierdoor een hiaat in mijn kennis van Oracle software kunnen opvullen. Doordat ik een aantal keren tegen de grenzen van PL/SQL aanliep, moest ik me verdiepen in de implementatie van Java classes in een Oracle database.

Het schrijven van dit verslag moest ik combineren met een detacheringklus in Utrecht. Dit kostte veel energie, waardoor het afronden van dit onderzoek langer duurde dan ik gehoopt had. Ik ben blij dat ik het resultaat nu kan presenteren.

Tenslotte wil ik de collega’s in Groningen, Ingrid Jansen, Sybrand Houtsma, Jonathan Engberts en Erik Loosman danken voor hun hulp en waardevolle informatie.

1 Inleiding

Het onderzoek waarvan hier het resultaat wordt gepresenteerd is de afronding van de Master opleiding Software Engineering aan de Universiteit van Amsterdam.

Het onderzoek vond plaats in de periode oktober 2009 – maart 2010 bij het Applicatie Support team van Ordina Oracle Solutions. Ordina is een beursgenoteerd bedrijf met ca. 3300 werknemers, dat zich profileert als kennisleverancier op het gebied van financiële dienstverlening, overheid, zorg en enkele segmenten binnen de industriemarkt [Bron: jaarverslag 2010 – http://jaarverslag.ordina.nl]. De auteur is in dienst als software engineer bij Ordina Oracle Solutions, een onderdeel van Ordina ICT B.V., en maakt zelf geen deel uit van het Applicatie Support team, maar is meestal gedetacheerd bij klanten.

Het Applicatie Support team van Ordina Oracle Solutions doet het beheer van applicaties die gebaseerd zijn op Oracle technologie. Voor een groot deel zijn dit applicaties van klanten. Daarnaast zijn er nog enkele applicaties die binnen de organisatie zelf worden gebruikt en zijn ontwikkeld. Het zijn allemaal relatief kleine applicaties, die samen door één team worden beheerd. Het beheer van de grotere Oracle-applicaties wordt door speciale teams uitgevoerd.

Het beheer dat het Applicatie Support team van Ordina Oracle Solutions uitvoert bestaat uit een aantal verschillende taken. Dit wordt weerspiegeld in vier typen meldingen die gehanteerd worden in het issuemanagement-systeem dat binnen het team gebruikt wordt:
- incident
- wijziging
- probleem
- reguliere wijziging

Incidenten zijn meldingen die door de gebruikers worden ingediend wanneer er iets met de applicatie aan de hand is. Dit kan worden opgelost door bijvoorbeeld de applicatie opnieuw te activeren, door corrupte data te muteren of door de gebruiker extra informatie te geven. In deze gevallen is geen wijziging aan de software noodzakelijk. Dat is ook het geval met reguliere wijzigingen: dit betreft zaken als het aanmaken van nieuwe gebruikers of het up-to-date houden van inrichtingsgegevens. Wanneer er wel een wijziging in de software nodig is, maakt de medewerker van het Applicatie Support team een nieuwe melding aan van het type probleem of wijziging. Een wijziging heeft betrekking op een gewenste wijziging in het ontwerp. Een probleem heeft betrekking op zaken die niet werken conform het ontwerp (defecten).

Deze werkwijze laat zien dat het beheer van software verschillende activiteiten omvat. In het vervolg van deze scriptie wordt de term onderhoud gereserveerd voor activiteiten die betrekking hebben op wijzigingen aan de source code. Voor het geheel van activiteiten, het onderhoud in de ruime zin van het woord, wordt de term beheer gebruikt.

2 Probleemstelling

2.1 Inleiding

In het verleden is gebleken dat het in beheer nemen van een applicatie door een organisatie die niet bij de ontwikkeling betrokken is geweest, risico's met zich meebrengt. Het blijkt moeilijk om in te schatten wat de benodigde beheerinspanning voor een onbekend systeem zal zijn. Daardoor is het moeilijk om resources te plannen. Wanneer een de beheersinspanning te optimistisch wordt ingeschat leidt dit tot ongewenste budgetoverschrijdingen.

De oorzaak hiervan is dat de gangbare applicatie-kenmerken, die gebaseerd zijn op database-volume en functiepunten, geen goede indicatoren blijken te zijn. Bij de intake van een nieuwe applicatie vindt vaak wel een code review plaats, maar dit is niet gestandaardiseerd. De kwaliteit van de code review is dus afhankelijk van de inzichten van degene die de review doet en de hoeveelheid tijd deze daarvoor ter beschikking krijgt.

Een mogelijkheid om dit proces te kunnen verbeteren is het toepassen van automatische code reviews. De vraag die in dit onderzoek behandeld zal worden welke toegevoegde waarde het uitvoeren van automatische code reviews zal hebben.

2.2 Outsourcing van softwareonderhoud

2.2.1 Outsourcing vanuit het perspectief van de beheerorganisatie

De context van dit onderzoek is outsourcing. Outsourcing van het onderhoud zorgt voor extra complexiteit omdat bij het beheer van de software meer dan één partij betrokken is. Communicatie, met name uitwisseling van kennis, speelt hierin een belangrijke rol. Deze problematiek wordt gevoeld zowel bij de organisatie die de software gebruikt als bij de organisatie bij wie het onderhoud is uitbesteed (2). Dit onderzoek wordt uitgevoerd vanuit het perspectief vanuit het laatstgenoemde type organisatie.

2.2.2 Continuïteit tussen ontwikkel- en onderhoudsteam

Een van de problemen bij het beheer is dat tijdens de levenscyclus van een applicatie de aanwezige kennis in het onderhoudsteam afneemt. Het gevolg is dat de architecturele integriteit afneemt totdat het systeem niet meer te onderhouden is. Aanvankelijk kunnen nog relatief grote wijzigingen kunnen worden aangebracht, terwijl in een later stadium alleen nog kleine wijzigingen mogelijk zijn (3). Raylich en Bennett benadrukken de noodzaak van continuïteit tussen ontwikkelteam en onderhoudsteam voor software evolutie. Grondige kennis van architectuur en ontwerp zijn noodzakelijk om omvangrijke aanpassingen te kunnen doen. Zij stellen dat outsourcen van software-evolutie om deze reden erg moeilijk is.

2.2.3 Belang van code assessments bij overdracht naar een beheersorganisatie

Waar dit toch gebeurt zal bij de overdracht van een applicatie van de ene naar de andere organisatie dus grondige kennisoverdracht moeten plaatsvinden. Dit is niet alleen functionele kennis en architectuurkennis, maar ook technische kennis. Een belangrijke factor hierin is op welke manier tijdens de bouw en eventueel onderhoud daarna aandacht is besteed aan de onderhoudbaarheid.

Een van de acties die bij de acceptatie van software door een beheerorganisatie moeten worden uitgevoerd om te controleren of de technische kwaliteit hoog genoeg is, is het uitvoeren van een automatische code review.

Aspecten van de code die volgens (4) onderzocht moeten worden zijn:

- omvang van de totale applicatie, de programma's en de modules
- complexiteit van de applicatie (Halstead en McCabe)
- toepassen coding standards
- softwarearchitectuur (helderheid van opbouw en structuur, voldoet de uitwerking aan de architectuur)

Een geautomatiseerde code review kan volgens (4) de volgende zaken betreffen:
- omvang en complexiteit
- coding standards
- aanwezigheid van commentaar
- afwezigheid van dode of gedupliceerde code

2.3 Te onderzoeken applicaties

Het Applicatie Support team doet het beheer aan applicaties die zijn gebaseerd op Oracle technologie. Dat zijn applicaties die gebruik maken van een Oracle (relationele) database, waarin gegevens persistent worden opgeslagen. Deze gegevens kunnen worden getoond of gemuteerd door client software of door interfaces met andere systemen. De client software kan gebaseerd zijn op Oracle technologie, maar er zijn ook andere clients gangbaar, zoals Java of Progress.

In de database kunnen controles worden geprogrammeerd om data-integriteit te bewaken. Daarnaast kan in de database logica worden vastgelegd en uitgevoerd voor het batchmatig verwerken van data en voor het bewaken en uitvoeren van business rules. Hiervoor is een domeinspecifieke taal beschikbaar, PL/SQL. Dit is een procedurele taal, die verwant is aan ADA, met embedded SQL. Hoewel PL/SQL een domeinspecifieke taal is, is de syntax zo uitgebreid dat het de kenmerken van een general purpose taal heeft. Sinds enkele jaren ondersteunt Oracle ook het opslaan en uitvoeren van Java code in de database.

Oracle voorziet ook in client software voor het benaderen van de database: Oracle Forms en Oracle Reports. In Oracle Forms kunnen gegevens uit de database worden geraadpleegd of gemuteerd. Het ophalen en muteren van gegevens gebeurt met SQL. Daarnaast kan er logica aanwezig zijn voor de schermafhandeling, zoals het conditioneel muteerbaar maken van invoervelden, het tonen van meldingen bij controle op de invoer en het starten van een batchproces d.m.v. een button. Voor deze logica wordt ook PL/SQL gebruikt. Dit is een andere versie dan de PL/SQL die in de database gebruikt wordt. Het verschil bestaat voornamelijk in het gebruik van speciale built-ins. De PL/SQL code wordt aangeroepen door middel van triggers, die afgaan op events die in Oracle Forms kunnen worden gedefinieerd (bijvoorbeeld een actie nadat een SQL-query is uitgevoerd of een actie bij het klikken op een button). Oracle Reports is een ander product van Oracle, maar berust wat het gebruik van SQL en PL/SQL betreft op hetzelfde principe. Hoewel reports bedoeld zijn om gegevens te tonen is het ook mogelijk om gegevens weg te schrijven naar de database. Dit gebeurt door middel van PL/SQL code die wordt uitgevoerd bij het afgaan van een trigger-event. Voorbeelden hiervan zijn het vastleggen of, of op welk tijdstip een rapportage is uitgevoerd, of tijdelijke opslag van het resultaat van een berekening of complexe SQL-query om performance redenen.

In dit onderzoek beperken we ons tot het onderhoud aan de PL/SQL component van applicaties. Dit betekent dat alleen Oracle Forms applicaties worden onderzocht. Oracle Reports wordt meestal gebruikt in combinatie met Oracle Forms, dus dit product wordt in dit onderzoek beschouwd als een component van een Oracle Forms applicatie. Applicaties met alleen een PL/SQL-component worden ook wel klassieke Oracle Forms applicaties genoemd. Applicaties met een significante component die is gebouwd in een andere taal, zoals Java of C++, vallen buiten de scope van dit onderzoek (zie § 3.3).

Door de beperking tot één taal ontstaat een groep van applicaties die met elkaar vergeleken kunnen worden en wellicht ook met andere groepen applicaties. Binnen de groep kunnen er wellicht verschillen in programmeerstijl of complexiteit worden waargenomen. Doordat er maar één taal wordt onderzocht is het effect dat dezelfde logische constructie in de ene taal meer regels code vereist dan in

de andere geëlimineerd. De tweede reden om het onderzoek te beperken tot één taal is van praktische aard, zodat voor de analyse van alle applicaties één analysetool kan worden gebruikt (zie hoofdstuk 4).

Bij dit onderzoek worden de andere componenten van Oracle Forms en Oracle Reports, bijvoorbeeld het datamodel of Java applets, buiten beschouwing gelaten. De reden hiervoor is dat uit ervaring is gebleken dat de basisfunctionaliteit van een invoerscherm (het tonen van database-velden, het implementeren van constraints op tabellen) met behulp van speciale tools (Oracle Forms Builder, Oracle Designer) vrij snel gerealiseerd kan worden en dat de meeste effort nodig is voor het implementeren van business rules en het bijbehorende gedrag van het scherm. Dat laatste gebeurt voornamelijk in PL/SQL. Verwacht mag worden dat het onderhoud dus voornamelijk in de PL/SQL-laag plaatsvindt.

Momenteel wordt de Oracle Forms technologie steeds minder toegepast en neemt het aanbod nieuwe PL/SQL programmeurs af. In sommige beheerorganisaties worden klassieke Oracle Forms applicaties al tot de legacy gerekend. Toch zijn er bedrijfskritische applicaties gebouwd met deze technologie. Te verwachten is dat het onderhoud aan deze applicaties steeds meer een specialiteit wordt en dat bedrijven dit daarom gaan uitbesteden aan gespecialiseerde organisaties zoals Ordina. Voor deze organisaties is het daarom belangrijk om bij de intake een gestandaardiseerde kwaliteitsbepaling te gebruiken.

2.4 Kwaliteitscriteria

Er zijn in het verleden veel criteria (metrieken) ontworpen om de kwaliteit van software te beschrijven. In dit onderzoek is gekozen voor de criteria van het SIG maintainability model, die zijn opgesteld door de Software Improvement Group (www.sig.eu). In zijn afstudeerscriptie stelt Frank Oppedijk dat dit model nog geen wetenschappelijke validatie heeft ondergaan, maar dat het zijn commerciële waarde bewezen heeft in vele systeem-assessments die de SIG heeft uitgevoerd (5). Dit onderzoek wil een bijdrage leveren aan de validatie van dit model.

Het voordeel van dit model boven een enkel getal als maat voor de onderhoudbaarheid, zoals de Maintainability Index, is dat het gebruik van afzonderlijke criteria een root cause analyse mogelijk maakt.

Het model beschrijft een viertal aspecten van software kwaliteit:
-analyseerbaarheid
-wijzigbaarheid
-stabiliteit
-testbaarheid
Om deze te kunnen bepalen worden moeten vijf basiscriteria worden bepaald:
-volume
-complexiteit
-duplicaten (redundantie)
-unit-grootte
-unit tests

Omdat niet voldoende informatie aanwezig was over unit tests bij de te onderzoeken applicaties is dit criterium niet meegenomen in het onderzoek. De andere vier wel. Oppedijk laat in zijn onderzoek dit criterium eveneens achterwege.

In het SIG maintainability model worden de gevonden waarden voor de basiscriteria vertaald naar een score op een 5-puntsschaal van -- tot ++. Volgens een mapping-tabel (zie Tabel 1) worden deze scores vertaald naar een score voor elk van de vier kwaliteitsaspecten.

volume	complexiteit	duplicaten	unit-grootte	unit tests	kwaliteitsaspect
x		x	x		analyseerbaarheid
	x	x			wijzigbaarheid
				x	stabiliteit
	x		x		testbaarheid

Tabel 1 Vertaling basiscriteria naar kwaliteitsaspecten, ontleend aan (5)

2.5 Hypothese

De hypothese die we willen onderzoeken luidt:

De criteria van het SIG maintainability model voldoen als beschrijving van de kwaliteit van Oracle Forms applicaties.

Deze formulering is nog vrij algemeen. In § 3.8 zal deze verder worden uitgewerkt.

3 Onderzoeksmethode

3.1 Onderzoeksmateriaal

Een softwareapplicatie bestaat uit softwarecomponenten geschreven zijn in een computertaal. Hoewel computertalen formele talen zijn, bieden ze een zekere vrijheid aan de programmeur om een gegeven probleem op te lossen. PL/SQL heeft de kenmerken van een general purpose taal, waardoor deze vrijheid relatief groot is. Door het toepassen van coding standaarden en het gebruiken van tools wordt deze vrijheid beperkt, zodat binnen een applicatie de componenten een zekere uniformiteit bezitten. De mate waarin programmeurs coding standaarden hebben toegepast kan een maat zijn voor de kwaliteit van de software.

De vier criteria van het SIG Maintainability model die onderzocht worden betreffen talige constructies:
Volume: regels tekst
Complexiteit: voorkomen van bepaalde uitdrukkingen
Duplicaten: identieke tekstpassages
Unit-grootte: teksteenheden

Het SIG Maintainability model abstraheert dus van de complexiteit van het domein. De vier criteria zeggen ook niet direct iets over het gebruik van coding standaarden.

3.2 Empirische methode

Om de bruikbaarheid van de criteria uit het SIG Maintainability model te toetsen is gekozen voor een empirisch onderzoek. Gezien het feit dat het onderzoeksmateriaal voor een deel bepaald is door menselijke vrijheid en creativiteit mag niet verwacht worden dat er verbanden gevonden worden met een zelfde precisie als bij natuurwetenschappelijk onderzoek.

De onderzochte software is bestaande software, dus niet speciaal voor dit onderzoek geschreven. Hierop is voor zover bekend tijdens de ontwikkelfase geen automatische codeanalyse uitgevoerd. In deze opzichten verschilt dit onderzoek van een laboratoriumonderzoek.

De methodiek die gevolgd wordt is dus die ook gebruikt wordt in de sociale wetenschappen. We mogen wel verwachten dat een zelfde onderzoek in een soortgelijke populatie dezelfde resultaten zal opleveren.

3.3 Onderzochte populatie

Er is gebruik gemaakt van een doelgerichte steekproef. De populatie van te onderzoeken applicaties is beperkt tot applicaties die volgens dezelfde technologie gebouwd zijn. Hierdoor zijn invloeden van eigenaardigheden van computertalen, zoals verbositeit, object-georiënteerd of procedureel uitgesloten. Daarnaast is van elke applicatie maar één versie onderzocht. Hiermee is onderlinge afhankelijkheid uitgesloten.

De te onderzoeken applicaties moesten al enige tijd in onderhoud zijn, zodat er voldoende informatie aanwezig was over de beheersinspanning. De beheersinspanning moest voor alle applicaties op dezelfde manier zijn bijgehouden. Deze criteria hebben ertoe geleid dat grote applicaties buiten de scope van het onderzoek zijn gebleven. Toepassing van deze criteria leidde tot een negental applicaties die konden worden onderzocht.

Het doel waarvoor een applicatie is ontworpen is geen selectiecriterium geweest.

3.4 Grootheden

De abstracte grootheden uit het SIG Maintainability model zijn voor elke applicatie als volgt gemeten:
- LOC: Het aantal regels code als maat voor het volume, niet gecorrigeerd voor lege regels en commentaarregels
- Complexiteit: Het percentage regels code van units met een cyclomatische complexiteit groter dan een gegeven minimum (10, 20, 50). Deze grootheid wordt weergegeven als percentage van het totaal aantal regels code.
- Duplicaten: Het percentage regels dat meer dan één keer voorkomt in de applicatie, geteld wanneer deze deel uitmaken van set van meer dan 6 regels die elders ook voorkomen. In het zoekpatroon wordt niet gekeken naar commentaar en layout. Deze grootheid wordt weergegeven als percentage van het totaal aantal regels code.
- Unit grootte: Het percentage regels code van units met een lengte (LOC) groter dan een gegeven minimum (10, 50, 100). Deze grootheid wordt weergegeven als percentage van het totaal aantal regels code. In dit onderzoek is een Oracle PL/SQL unit gedefinieerd als een procedure, functie of een trigger (forms en database). Units die zijn gedeclareerd binnen een andere unit (subunits) worden apart geteld.

Voor het bepalen van de cyclomatische complexiteit is het nodig om de afzonderlijke units uit de stored packages en Forms PL/SQL libraries te bepalen. Immers, de cyclomatische complexiteit heeft betrekking op een afzonderlijke control flow in een programma (procedure, functie of trigger). Dit wordt verder uitgewerkt in hoofdstuk 4.

In de tweede plaats worden de gangbare beoordelingscriteria voor Oracle Forms applicaties onderzocht. Dat zijn:
- Aantal tabellen
- Aantal kolommen
- Aantal functiepunten: Hierbij kan gekeken worden naar het totale aantal zelfstandige modules, of alleen naar de client-modules (forms en reports).

3.5 Bepaling ISO9126 subkarakteristieken

De gemeten grootheden kunnen volgens het SIG-maintainability model worden vertaald naar software kwaliteitscriteria. We kunnen de hypothese uit § 2.5 ook zo formuleren: Het SIG-maintainability model levert een duidelijk beeld op van de gemeten verschillen tussen de applicaties. Ook deze formulering is nog vaag en zal daarom per component van het model (zie § 3.8) verder worden uitgewerkt.

3.6 Gegevens uit de meldingendatabase

Bij Ordina Oracle Solutions is de informatie over de beheersinspanning voor de applicaties vastgelegd in een issuemanagementsysteem. Dit is een Oracle database waarin meldingen van een gegeven type worden vastgelegd. Alle te onderzoeken applicaties zijn hierin opgenomen. Voor dit onderzoek is van belang dat bij elke behandelde melding de bestede tijd in uren is vastgelegd.

Per applicatie zijn alle meldingen geselecteerd van het type
- incident,
- wijziging,
- probleem en
- reguliere wijziging,

met status “afgesloten” en waarbij meer dan 0 uren bestede tijd is vastgelegd. Overige meldingtypes zijn niet in het onderzoek meegenomen: in de eerste plaats omdat daarvan voor de geselecteerde applicaties relatief weinig uren geregistreerd waren en in de tweede plaats omdat deze meldingtypes weinig informatie geven over de onderhoudsinspanning.

Het aantal meldingen en het aantal bestede uren is geaccumuleerd per jaar, per applicatie, gerekend vanaf de eerste datum waarop (voor de betreffende applicatie) een melding (ongeacht type) is ingediend. In dit onderzoek wordt naar de totale beheerinspanning gekeken, dat wil zeggen alle meldingen en bestede uren geaccumuleerd van de eerste datum waarop een melding is ingediend tot een peildatum (20 april 2010).

Om de totale beheers-/onderhoudsinspanning te kunnen gebruiken nemen we aan dat het aantal meldingen/uren in de tijd toeneemt tot een asymptotische waarde. Bij dit onderzoek is aangenomen dat dit voor alle applicaties het geval is. Om deze aanname te kunnen controleren is achteraf ook het tijdstip van indienen in beschouwing genomen.

3.7 Validatie van de componenten van het model

Voor validatie van de gebruikte criteria is onderzocht of er correlaties konden worden gevonden tussen de gemeten code-eigenschappen en de gegevens uit de meldingendatabase. De volgende correlaties worden onderzocht:
1. De correlatie tussen het totaal aantal bestede uren en de gemeten grootheden. Dit is een maat voor de benodigde onderhoudsinspanning/beheersinspanning.
2. De correlatie tussen de aantallen meldingen waarop uren zijn geboekt en de gemeten grootheden. Dit zegt iets over de belasting van de beheersorganisatie en kan een maat zijn voor de tevredenheid/ontevredenheid van de klant als gevolg van codekwaliteit.
3. De correlatie tussen het gemiddeld aantal uren per melding en de gemeten grootheden. Dit is een maat voor de onderhoudbaarheid van de software.

De correlaties worden onderzocht per grootheid, per type melding en voor alle meldingen samen (de totale beheersinspanning).

De aard van de mogelijke verbanden (lineair, exponentieel, logaritmisch etc.) is niet a priori bekend. De populatie die voor het onderzoek bruikbaar was bleek niet groot te zijn - 9 applicaties – en bevatte ook geen grote applicaties (zie §3.3). Daarom is het niet waarschijnlijk dat de data normaal verdeeld zijn. Om deze reden is afgezien van het bepalen van de Spearman correlatiecoëfficiënt, omdat deze bepaling veronderstelt dat de data normaal verdeeld zijn en dat het te onderzoeken verband lineair is. Er is gekozen voor het bepalen van de Kendall tau. Deze bepaling is gebaseerd op rangnummers in plaats van de gemeten waarden zelf. Het getal geeft de verhouding weer tussen het aantal concordante paren en het aantal disconcordante paren.

3.8 Hypothese

De hypothese uit § 2.5 kan nu verder worden uitgewerkt. De meer specifieke hypothese is dat er een correlatie bestaat tussen de gemeten code-eigenschappen en de uren respectievelijk aantallen meldingen in de meldingendatabase.

Hierbij moet worden opgemerkt dat het vinden van een correlatie niet betekent dat er een direct causaal verband is. Het is mogelijk dat de variatie in beide grootheden een gemeenschappelijke oorzaak heeft, of dat het verband toevallig is. Het toetsen van de hypothese gebeurt door toepassen van het falsificatieprincipe.

3.9 Toetsing van de hypothese

De falsificatie wordt gedaan door van de berekende Kendall tau de significantie te bepalen. Dit gebeurt door middel van een t-toets met een acceptatiegrens van 5%. De nulhypothese waartegen getest wordt luidt: de Kendall tau heeft de waarde 0, dat wil zeggen: er is geen correlatie tussen beide grootheden. (Dit is het omgekeerde van de onderderzoekshypothese) Wanneer de t-waarde kleiner is dan 0,05 dan wordt de nulhypothese verworpen. In dat geval is er wel sprake van een correlatie. De onderzoekshypothese voor het te onderzoeken criterium is dus in dit geval niet gefalsifieerd.

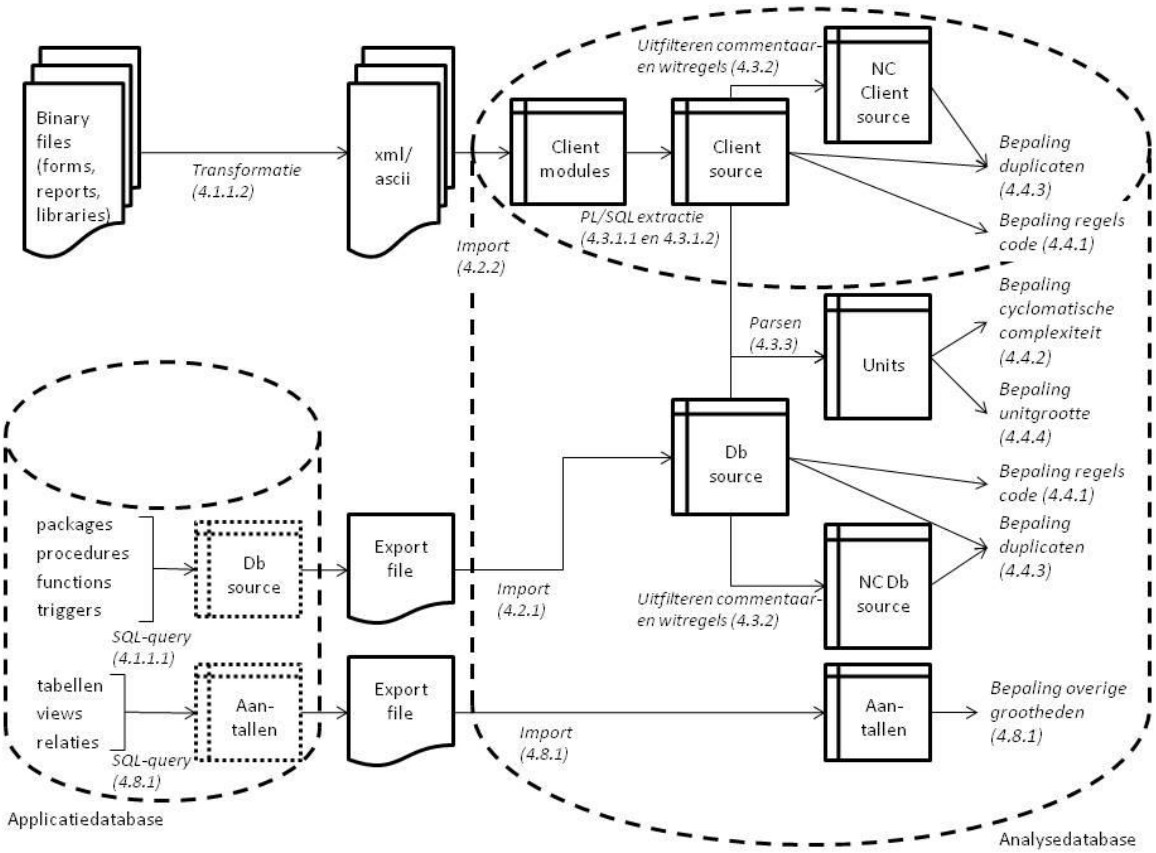
4 Onderzoek

Het onderzoek bestond uit de volgende stappen:
1. Het uitvoeren van statische codeanalyse
2. Het toepassen van het SIG-maintainability model op de verzamelde data
3. Het verzamelen van gegevens over de beheersinspanning uit de meldingendatabase
4. Het bepalen van correlaties tussen de gegevens die verkregen zijn uit de statische codeanalyse en de gegevens uit de meldingendatabase
5. Het bepalen van correlaties tussen overige applicatie-eigenschappen en de gegevens uit de meldingendatabase

In Figuur 1 is schematisch weergegeven welke stappen zijn uitgevoerd bij de statische codeanalyse. Bij elke stap staat het paragraafnummer waarin deze wordt beschreven. Voor dit doel is in de analysedatabase een tool gebouwd, deels in PL/SQL en deels in Java. De paragrafen 4.2 t/m 4.4 vormen een technische beschrijving van deze tool.

De centrale datastructuur is die van de tabel Db source, waarin elke regel code van de database-programmatuur afzonderlijk is vastgelegd (§ 4.2.1). Dit is identiek aan de wijze waarin Oracle de database-programmatuur vastlegt. De structuur van de Client source tabel is hiervan afgeleid (zie § 4.3.1).

De meest complexe stappen waren de extractie van PL/SQL-code uit de client modules (§ 4.3.1) en het bepalen van de afzonderlijke units uit de client- en database source code (§ 4.3.3).



Figuur 1 Implementatie van het onderzoek

4.1 Statische codeanalyse

Voor het de statische codeanalyse moesten de volgende acties worden uitgevoerd:
1. Extractie van de ruwe data
2. Importeren van de gegevens in een Oracle database
3. Bewerken van de gegevens
4. Uitvoeren van de analyses

4.1.1 Extractie van de ruwe data

4.1.1.1 Database software

Alle source code die is opgeslagen in de database, stored procedures, functions, packages en triggers, is met behulp van een SQL-query op ALL-SOURCE view opgehaald. Omdat deze view de gewenste datastructuur heeft - per regel de object-owner, de objectnaam, het type object en een tekstregel – is het resultaat van deze query in een tijdelijke tabel opgeslagen en van deze tabel een export gemaakt (dmp-file). Dit is voor elke applicatie afzonderlijk uitgevoerd.
Eén applicatie was geïnstalleerd op een Oracle 8 database. Hiervan kon geen export worden gemaakt in de vorm van dmp-file. Deze gegevens zijn geëxporteerd in de vorm van een ascii-file met insert-statements.

4.1.1.2 Client software

Alle source code die is opgeslagen in binary files op het filesysteem moest eerst geconverteerd worden:
- forms (fmb) en reports (rdf) naar xml,
- pl/sql libraries (pll) naar ascii (pld).
Hierbij is gebruik gemaakt van de utilities die Oracle hier utilities beschikbaar voor heeft (iff2xml90, rwconverter en ifcmp90 voor resp. forms, reports en libraries).

4.2 Importeren van de gegevens in een Oracle database

4.2.1 Database software

De database software kon rechtstreeks geïmporteerd worden uit de gemaakte exportbestanden.

4.2.2 Client software

De geconverteerde bestanden werden ingelezen met behulp van een speciaal hiervoor geschreven database procedure. Deze procedure maakt gebruik van Oracle procedure dbms_lob.loadfromfile. De bestanden werden als geheel ingelezen in een tabel, d.w.z. één rij per bestand. De inleesprocedure heeft de applicatienaam als inputparameter, deze moest bij aanroep met de hand worden ingevoerd. Hierdoor werd de applicatienaam gekoppeld aan de ingelezen bestanden.

4.3 Bewerken van de gegevens

4.3.1 Extractie van PL/SQL-tekstregels uit client files

4.3.1.1 Extractie uit xml-bestanden

Van de xml-bestanden werden alle nodes met PL/SQL-code (ProgramUnitText, TriggerText bij forms en programUnits/function/textSource bij reports) omgezet naar tekst. Hierbij was het in het geval van Forms nodig om alle geconverteerde entity references (<, >, newline, quote) terug te converteren. Hierdoor kon elke PL/SQL-tekstregel afzonderlijk worden opgeslagen in een tabel met source code regels. Dit is een tabel met dezelfde structuur als die vooor database-PL/SQL.
Voor trigger tekst moest nog een extra actie worden uitgevoerd. Forms-triggers kunnen op verschillende niveaus voorkomen (Form, Block, Item). Hierdoor hebben ze geen unieke naam. Dit is opgelost door door de block- en/of itemnaam aan de triggernaam toe te voegen, afhankelijk van het niveau van de trigger. Aangezien Forms-triggers geen speciale header hebben is met het oog op het bepalen van afzonderlijke units (§ 4.3.3) een dummy header en een begin en een end keyword toegevoegd zodat de code voldoet aan dezelfde syntax als database-triggers. Vanwege de omvang van sommige program units was het niet mogelijk om alle tekstbewerkingen in PL/SQL uit te voeren. Deze functionaliteit moest worden geïmplementeerd door een Java klasse.

4.3.1.2 Extractie uit pld-bestanden

Voor pld-bestanden was geen verdere conversie nodig. Deze bestanden konden regel voor regel worden ingelezen in de tabel met client-source regels.

4.3.2 Uitfilteren commentaar- en witregels (voor duplicaten)

Met het oog op het bepalen van duplicaten zijn de tekstregels ook apart opgeslagen zonder commentaar (embedded comment, end line comment) en witregels, met hetzelfde regelnummer.

4.3.3 Bepalen afzonderlijke units

Met het oog op de bepaling van de cyclomatische complexiteit zijn alle units en alle regels code bij de units apart opgeslagen. Hiervoor werden de afzonderlijke regels per client-bestand en database-programma weer samengevoegd tot een teksteenheid. Deze teksteenheid werd d.m.v. een parser geconverteerd naar losse units. Deze parser is m.b.v. Antlr gegenereerd en speciaal voor dit onderzoek gemaakt. De hiervoor benodigde PL/SQL-grammatica is gecreëerd d.m.v. reverse engineering uit de beschikbare programma's. Daarom is het geen volledige grammatica, maar toegesneden op het herkennen van afzonderlijke units (eiland parsing, zie bijlage A). De gegenereerde Java-code is in de database ingeladen en geïntegreerd met de analysetool.

Een klein aantal programma's in de database bleek te zijn gewrapped. Deze konden niet worden geanalyseerd. Het betreft bijvoorbeeld functionaliteit voor authenticatie en autorisatie. Dit is aangetroffen in slechts één applicatie, applicatie D. Het aantal regels niet-geanalyseerde code bleek te verwaarlozen ten opzichte van het totaal.

4.4 Uitvoeren van de analyses

4.4.1 Bepaling volume

Voor het bepalen van het volume van een applicatie zijn alle afzonderlijke regels PL/SQL-code geteld. De waarden voor de database-software en voor de client-software werden bij elkaar opgeteld. De dummy regels die zijn toegevoegd om Forms triggers te kunnen parsen zijn niet meegeteld.

4.4.2 Bepaling cylcomatische complexiteit

Bepaald is de extended cyclomatische complexiteit, d.w.z. dat logische operatoren (and en or) zijn meegeteld. Deze operatoren komen veelvuldig voor in SQL-statements. SQL-statements zijn in dit onderzoek niet meegeteld, tenzij het sql-statement binnen een begin-end block stond (een verdere verfijning is niet toegepast, omdat dit in de wat oudere PL/SQL-programmatuur niet gebruikelijk is). De volgende PL/SQL-expressies zijn meegeteld: if, else, elsif, while, for en when (in case en exceptions). Er is één uitzondering gemaakt: de when in when others is niet meegeteld omdat deze exception in principe op elk punt van de control flow kan optreden.
De logische operator and die deel uitmaakt van een between expressie is niet meegeteld. Zie discussie in subparagraaf 6.1.2.1.3. Deze constructie komt echter in de praktijk niet veel voor in PL/SQL (wel in SQL).

4.4.3 Bepaling duplicaten

Er is gekozen is voor het zoeken naar duplicaten op basis van letterlijke tekst.
Hierbij is gezocht met behulp van een zoekpatroon. Dit zoekpatroon bestaat uit een gegeven aantal regels tekst, waaruit het volgende is verwijderd:
- end line comment (--)
- embedded comment (/* */)
- spaties
- tabs
- newline karakters, waarbij er één overgelaten wordt achter elke niet-lege string

Deze bewerking is ook uitgevoerd op de te doorzoeken tekst. Er is niet-hoofdlettergevoelig gezocht. Het zoekalgoritme start met een zoekpatroon van 6 regels onbewerkte tekst, waarna de bewerking wordt herhaald met een zoekpatroon van steeds 1 regel meer, totdat er geen match meer gevonden wordt of totdat het einde van de unit is bereikt. Het aantal regels originele tekst dat aan het langste patroon voldoet wordt geteld. Met unit wordt in deze context niet de kleinste programma-eenheid, maar de grootste programma-eenheid bedoeld.

Het zoeken naar duplicaten is uitgevoerd voor database-software en client-software apart. Dit is gebeurd onder de aanname dat de kans dat 6 regels code uit de database ongewijzigd voorkomt in de client-software en omgekeerd heel klein is.

4.4.4 Bepaling unitgrootte

Voor het bepalen van de unitgrootte zijn voor elke unit alle afzonderlijke regels bij elkaar opgeteld, inclusief commentaar en witregels. Voor een unit met subunits, zijn de regels van de subunits van het totaal van de unit afgetrokken. Packages, die per definitie alleen maar subunits bevatten, zijn niet meegeteld, tenzij ze een eigen begin-end block hadden (voor initialisatie).

4.5 Bepaling ISO9126 subkarakteristieken

Op basis van de meetwaarden zijn volgens het SIG-maintainability model scores bepaald en vertaald naar kwaliteitskarakteristieken. Dit is beschreven in § 2.4 en § 3.5.

4.6 Verzamelen gegevens uit de meldingendatabase

Voor de te onderzoeken applicaties (SLA's) zijn m.b.v. SQL alle meldingen geselecteerd
- met status "afgesloten",
- waarbij het aantal bestede uren groter is dan 0 en
- van meldingstype "wijziging", "reguliere wijziging", "probleem" en "incident".

Op basis hiervan zijn per applicatie en per meldingstype berekend:
- de som van de bestede uren,
- het aantal meldingen en
- (als afgeleid gegeven) het gemiddelde aantal bestede uren per melding

Per applicatie is tevens de tijd in jaren tussen de eerste melding en de laatste melding bepaald: dit is de beheerperiode.

Voor nadere analyse zijn eveneens per jaar, gerekend vanaf de datum van de eerste melding, per applicatie en per meldingstype de som van de bestede uren, het aantal meldingen en het gemiddelde aantal bestede uren per melding berekend. De resultaten hiervan zijn in dit onderzoeksverslag verder buiten beschouwing gebleven.

4.7 Bepaling van de Kendall-tau

De Kendall-tau is een maat voor de verhouding van het aantal concordante en het aantal disconcordante paren, wanneer paren meetwaarden (X, Y) met elkaar worden vergeleken. Er is sprake van concordante paren als X en Y beide toenemen (of afnemen), van disconcordante paren neemt Y af als X toeneemt of omgekeerd.

De formule voor de Kendall-tau luidt (bron: www.wikipedia.org):

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{\frac{1}{2}n(n-1)}$$

Wanneer de populatie meetwaarden gelijke X- of Y-waarden bevat moet een daarvoor een correctie worden uitgevoerd. De Kendall-tau heeft altijd een waarde tussen -1 en 1. Hoe dichter de absolute waarde bij 1 ligt, hoe sterker monotoon het verband is. Hoe dichter de absolute waarde bij 0 ligt hoe zwakker monotoon het verband is.

In dit onderzoek is voor de bepaling van de Kentall-tau gebruik gemaakt van een op Internet beschikbaar rekenprogramma van de K.U. Leuven: http://www.wessa.net/rwasp_kendall.wasp
In dit programma zijn alle te onderzoeken combinaties van grootheden ingevoerd. Voor elke combinatie zijn de Kentall-tau en de 2-zijdige p-waarde overgenomen.

4.8 Onderzoek van overige grootheden

4.8.1 Overige databaseobjecten

Om het aantal tabellen, kolommen en views te bepalen zijn een aantal queries gedaan op de ALL_TABLES, ALL_TAB_COLUMNS en ALL_VIEWS views in de data dictionary van elke applicatie. De resultaten zijn opgeslagen in tijdelijke tabellen. Deze zijn op dezelfde manier geëxporteerd en geïmporteerd in de analysedatabase als de database PL/SQL-code (§ 4.1.1.1 en § 4.2.1). Van de tellingen zijn ook de correlaties bepaald met de gegevens uit de meldingendatabase.

Ook de complexiteit van de database, weergegeven door het aantal 1-N-relaties en het aantal N-M-relaties is zijdelings meegenomen in dit onderzoek. De 1-N relaties zijn bepaald uit de ALL_CONSTRAINTS view: dit zijn alle constraints van het type 'R'. Voor de N-M relaties is het aantal intersectietabellen bepaald, aan de hand van de volgende criteria:
- er is een unieke index die alle foreign key kolommen van de tabel bevat
- alle kolommen van de tabel maken deel uit van een unieke index
- het aantal unieke indexen van de tabel <= 2.

4.8.2 Client modules

De aantallen van de client modules (zie § 4.1.1.2) konden bepaald worden uit de tabel met geïmporteerde client modules. Van deze aantallen is ook de correlatie bepaald met de gegevens uit de meldingendatabase.

4.8.3 Gini-coëfficiënten

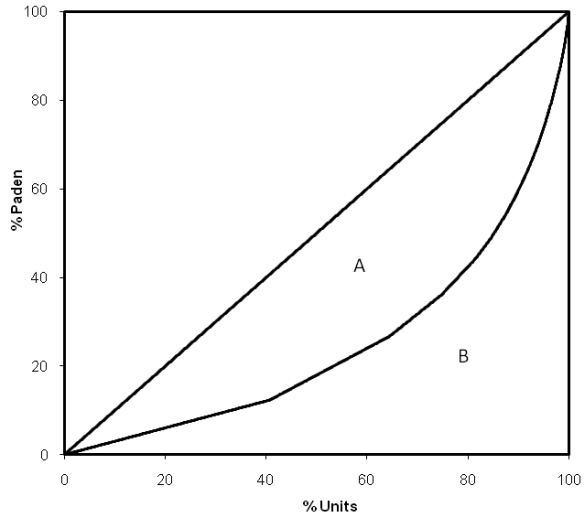
Als alternatief voor de bepaling van de unitgrootte en de cyclomatische complexiteit is ook nog de verdeling van deze grootheden over de software onderzocht. Hiervoor zijn per applicatie alle gemeten waarden in een spreadsheet gezet en gesorteerd naar grootte. Vervolgens zijn de waarden berekend als cumulatief percentage van het totaal aantal regels code. Met behulp van de formule van Brown is hiermee de Gini-coëfficiënt bepaald (bron: www.wikipedia.org):

$$G = |1 - \sum_{k=0}^{k=n-1} (X_{k+1} - X_k)(Y_{k+1} + Y_k)|$$

Hierbij is:

G de Gini-coëfficiënt
X de cumulatieve proportie van de variabele populatie (aandeel in de totale populatie)
Y de cumulatieve proportie van het variabele inkomen (aandeel in het totale inkomen)

In Figuur 2 is ter illustratie voor één van de onderzochte applicaties de verdeling van de cyclomatische complexiteit (het aantal paden per unit) over de units grafisch weergegeven. De Gini-coëfficiënt is hierbij de verhouding van de oppervlakte van het gebied tussen de schuine lijn en de curve (A) en de totale oppervlakte onder de schuine lijn (A + B). De schuine lijn geeft een exact gelijke verdeling van de grootheid over de individuen, in het geval van dit voorbeeld de units, weer.



Figuur 2 Voorbeeld Lorenzcurve voor de verdeling van de cyclomatische complexiteit over de units

Hoe dichter dus de Gini-coëfficënt bij 0 ligt, hoe gelijkmatiger de grootheid verdeeld is over het aantal regels code. Als de Gini-coëfficënt 1 is, dan is de grootheid alleen waargenomen bij één individu. Van deze Gini-coëffiënten is ook de correlatie bepaald met de gegevens uit de meldingendatabase.

5 Resultaten

5.1 Toepassing van het model: SIG ratings

In Tabel 2 worden de resultaten getoond van de toepassing van het SIG-maintainability model op de code van de applicaties. Gegevens over unit-tests waren niet voorhanden, zodat de stabiliteit niet bepaald is. Voor de criteria die zijn gebruikt voor het bepalen van de ratings zie bijlage C.

		Code-eigenschap				ISO 9126 subkarakteristiek			
Applicatie	Volume	Com-plexiteit	Dupli-caten	Unit-grootte	Unit-tests	Analyseer-baarheid	Wijzig-baarheid	Stabiliteit	Test-baarheid
A	+	-	--	--		-	--		--
B	+	--	--	--		-	--		--
C	o	-	--	--		-	--		--
D	o	--	--	--		-	--		--
E	++	o	--	--		-	-		-
F	-	-	--	--		--	--		--
G	+	-	--	--		-	--		--
H	+	o	--	--		-	-		-
I	o	-	--	--		-	--		--

Tabel 2 Resultaten toepassing SIG Maintainability model

Het SIG-Maintainability model laat zien dat er diversiteit is in de scores voor volume, dat alle applicaties relatief laag scoren op complexiteit en dat er geen onderscheid gemaakt kan worden op basis van de scores voor duplicaten en unitgrootte. Alle applicaties hebben voor duplicaten en unitgrootte de laagste score. Hierdoor geeft de mapping naar de ISO 9126 subkarakteristieken een vertekend beeld.

Afgezien van de volumemeting vallen alle scores relatief laag uit voor de onderzochte Oracle Formsapplicaties. Daarom is verder onderzocht of er een samenhang is met de verzamelde informatie uit de meldingendatabase.

5.2 Beheersinspanning

Het aantal uren dat per jaar besteed is is relatief laag: het komt overeen met ruwweg 0,75 fte. Dat komt omdat deze negen applicaties maar een deel van de beheeractiviteiten van het Applicatie Support team betreffen. Het team beheert nog meer applicaties, die niet in aanmerking kwamen voor dit onderzoek.

Tabel 3 toont voor elke applicatie de verdeling van de beheersinspanning over het type isssue.

	Periode	Totaal	Onderhoud		Incidenten		Regulier onderhoud	
Applicatie	Jaren	Uren	Uren	Percentage	Uren	Percentage	Uren	Percentage
A	5,0	228	38	17%	185	81%	5	2%
B	5,3	1354	629	46%	700	52%	25	2%
C	6,9	170	45	26%	118	69%	7	4%
D	5,8	1654	736	44%	862	52%	56	3%
E	2,0	134	14	10%	60	45%	60	45%
F	7,0	2642	2113	80%	527	20%	2	<1%
G	9,4	1907	138	7%	1501	79%	268	14%
H	4,5	236	53	22%	163	69%	20	8%
I	6,0	761	200	26%	416	55%	145	19%

Tabel 3 Verdeling van de beheersinspanning over het type melding

Hieruit blijkt dat applicatie F de grootste beheersinspanning heeft gekost. Het grootste deel daarvan is besteed aan onderhoud. Applicatie F heeft ook het grootste aantal regels PL/SQL (zie bijlage B, Tabel 14). Het verband tussen het aantal regels code en het aantal uren onderhoud wordt verder uitgewerkt in § 5.3.1.
Applicatie G is ook een van de applicaties die de meeste beheersinspanning heeft gekost, hiervan is het grootste deel besteed aan incidenten. De vraag is of dit samenhangt met de kwaliteit van de code. Uit de gemeten code-eigenschappen (zie bijlage B) blijkt applicatie G er niet uit te springen. Zie voor het verband tussen het aantal uren voor incidenten en de statische code-eigenschappen ook § 5.3.1.
Applicatie E heeft de minste beheersinspanning gekost, waarvan ook nog relatief weinig aan onderhoud is besteed. Omdat juist het meeste onderhoud verwacht mag worden in het begin van de beheerperiode, kan dit een aanwijzing zijn dat dit een goed onderhoudbare applicatie is. In elk geval is applicatie E in een aantal opzichten de kleinste applicatie (zie bijlage B).

Reguliere wijzigingen blijven bij de analyse buiten beschouwing omdat geen relatie verwacht mag worden met de eigenschappen van de code.

In Tabel 4 is de beheersinspanning voor de onderzochte applicaties weergegeven per type melding, met daarbij het aantal bestede uren per melding. Hieruit blijkt dat het aantal uren dat is besteed aan problemen relatief klein is ten opzichte van het aantal uren dat is besteed aan wijzigingen. Om deze reden worden alleen de totalen van wijzigingen en problemen verder in beschouwing genomen.

	Totaal onderhoud			Wijzigingen			Problemen			Incidenten		
Appli-catie	Mel-dingen	Best. uren	Uren/ Melding	Mel-dingen	Best. uren	Uren/ Melding	Mel-dingen	Best. uren	Uren/ Melding	Mel-dingen	Best. uren	Uren/ Melding
A	13	38	2,9	13	38	2,9	0	0	-	80	185	2,3
B	48	629	13,1	47	628	13,4	1	1	1,0	122	700	5,7
C	9	45	5,0	9	45	5,0	0	0	0,0	35	118	3,4
D	176	736	4,2	170	704	4,1	6	32	5,3	404	862	2,1
E	3	14	4,7	3	14	4,7	0	0	-	27	60	2,2
F	81	2113	26,1	76	2100	27,6	5	13	2,5	176	527	3,0
G	41	138	3,4	32	126	3,9	9	12	1,3	1384	1501	1,1
H	6	53	8,8	6	53	8,8	0	0	-	126	163	1,3
I	24	200	8,3	23	196	8,5	1	4	3,7	546	416	0,8

Tabel 4 Totale beheersinspanning per type melding

Uit Tabel 3 en Tabel 4 blijkt dat het aantal uren dat aan onderhoud is besteed varieert van 14 (applicatie E - over 2 jaar) tot 2113 (applicatie F - over 7 jaar). Dit zijn de kleinste en de grootste applicatie.
Voor applicatie D was het aantal onderhoudsmeldingen het hoogst (176), maar de gemiddelde oplostijd relatief laag (4,2 uur per melding). Uit bijlage B, Tabel 18 blijkt dat de verdeling van de cyclomatische complexiteit over de units bij applicatie D het meest gelijkmatig is. Dit is een aanwijzing dat deze applicatie relatief veel complexe units bevat. Uit Tabel 5 blijkt dat er inderdaad een verband is gevonden tussen het aantal meldingen en de verdeling (Gini-coëfficiënt) van de complexiteit over de units. Blijkbaar leidt dit tot relatief veel gemakkelijk te verhelpen wijzigingen.

De gemiddelde oplostijd van een onderhoudsmelding was voor applicatie F het hoogst (26,1 uur per melding) en voor de applicaties A en G het laagst (2,9 resp. 3,4 uur per melding).
Voor incidenten was de gemiddelde oplostijd voor applicatie B het hoogst (5,7 uur per incident) en voor applicatie I het laagst (0,8 uur per incident).
Deze uitersten met betrekking tot de gemiddelde oplostijd corresponderen niet met de uitersten van een van de gemeten statische code-eigenschappen. Dit komt wel overeen met de waarneming dat er voor de gemiddelde oplostijd bijna geen significante correlaties zijn gevonden. Zie volgende paragraaf.

5.3 Correlaties

5.3.1 Correlaties tussen statische code-eigenschappen en beheersinspanning

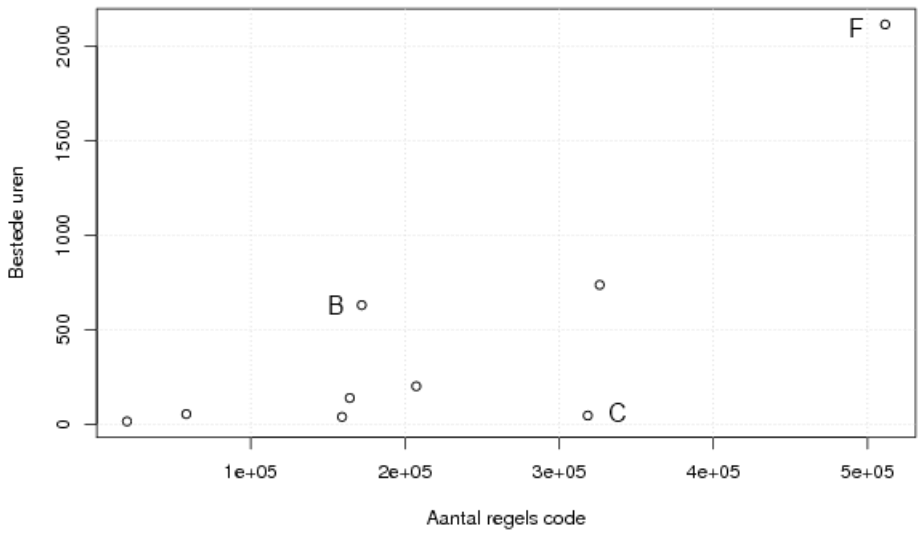
	uren		meldingen		uren per melding	
Criterium	kendall tau	significantie	kendall tau	significantie	kendall tau	significantie
LOC	**0,6667**	**0,016**	**0,6111**	**0,029**	0,2222	0,466
NCLOC	**0,6111**	**0,029**	**0,5556**	**0,048**	0,1667	0,602
%COM	-0,0556	0,917	0,1111	0,754	0,1667	0,602
%Duplicaten	0,2222	0,466	0,2778	0,348	0,0000	1,000
%LOC CC>10	0,4444	0,118	0,5000	0,076	0,1111	0,754
%LOC CC>20	0,3099	0,295	0,4789	0,093	-0,0845	0,834
%LOC CC>50	0,4789	0,093	**0,6480**	**0,021**	0,0282	1,000
G(Paden/Unit)	0,4444	0,118	**0,6111**	**0,029**	0,0000	1,000
%LOC Units>10	0,3889	0,175	0,4444	0,118	0,1667	0,602
%LOC Units>50	**0,6111**	**0,029**	**0,7778**	**0,005**	0,0556	0,917
%LOC Units>100	0,5000	0,076	**0,6667**	**0,016**	-0,0556	0,917
G(LOC/Unit)	0,3889	0,175	0,4444	0,118	0,1667	0,602

Tabel 5 Berekende correlaties voor totaal onderhoud

In bovenstaande tabel en de volgende tabellen zijn alle correlaties vet gedrukt die geen aanleiding geven om de hypothese dat er een correlatie is tussen de gemeten grootheid en de benodigde inspanning te verwerpen. In de kolom “Criterium” zijn de criteria die gebruikt worden in het SIG-Maintainability model vet gedrukt, de overige criteria zijn afgeleide criteria (NCLOC en %COM - percentage witregels en commentaarregels) en alternatieve criteria (Gini-coëfficienten voor cyclomatische complexiteit en unitgrootte).

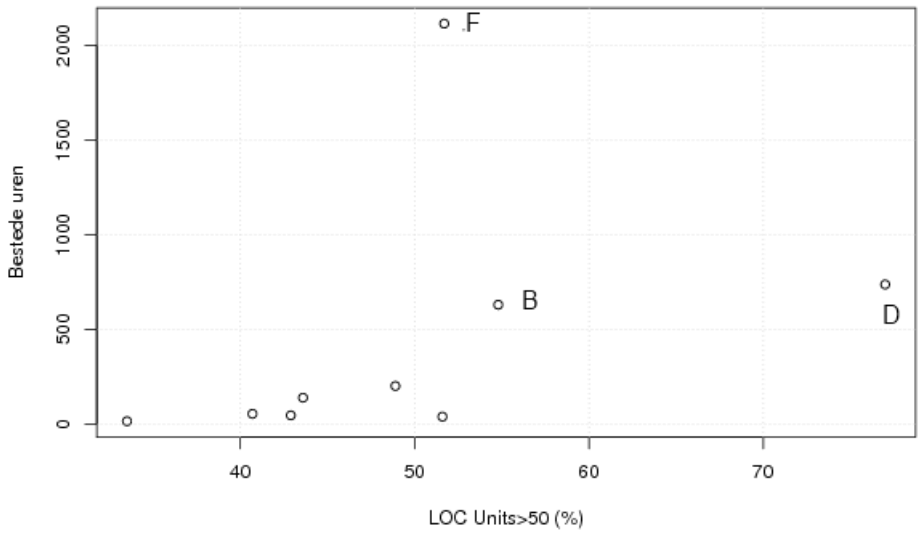
Uit Tabel 5 blijkt dat de hypothese voor het gemiddeld aantal uren per melding in alle gevallen verworpen is. Dat is ook het geval voor het percentage duplicaten.

Wat betreft de correlaties met het aantal bestede uren zijn is de hypothese verworpen voor alle gemeten grootheden, behalve het volume (LOC) en het percentage units dat groter is dan 50 regels. De mogelijke verbanden worden zichtbaar in Figuur 3 en Figuur 4.



Figuur 3 Totaal onderhoud: aantal regels code uitgezet tegen de bestede uren

Figuur 3 laat zien dat het aantal bestede uren hoger is naarmate de applicatie meer regels code bevat. De meetwaarden vertonen een mogelijk niet-lineair verband. Applicaties B en C vallen buiten dit patroon. Dit kan verklaard worden door de waarden voor de cyclomatische complexiteit die voor deze applicaties gevonden zijn: voor applicatie C vallen deze relatief laag uit, voor applicatie B juist relatief hoog. Deze waarnemingen laten zien dat de benodigde onderhoudsinspanning voor een applicatie inderdaad niet uit één factor kan worden verklaard. Gezien het feit dat voor de overige factoren geen significante correlaties zijn gevonden, mag wel worden aangenomen dat het aantal regels code de sterkste invloed heeft op de onderhoudsinspanning.



Figuur 4 Totaal onderhoud: percentage regels code in units >50 uitgezet tegen de bestede uren

Figuur 4 laat zien dat naarmate het percentage regels code dat zich bevindt in units groter dan 50 regels groter is ook het aantal het bestede uren groter is. Applicatie F past niet in dit patroon. Een verklaring hiervoor is dat niet de verdeling van de code over de units de oorzaak is van het hoge aantal bestede uren, maar de absolute grootte van de applicatie, zoals te zien is in Figuur 3. Mogelijk valt ook hier applicatie B buiten het patroon, om dezelfde reden als bij het totale aantal regels code.

Wat betreft de correlaties met het aantal meldingen kan de hypothese ook niet verworpen voor het percentage regels code van units met een cyclomatische complexiteit groter dan 50, voor de Gini-coëfficient van de verdeling van de cyclomatische complexiteit over de applicatie en het percentage units dat groter is dan 100 regels.

	uren		meldingen		uren per melding	
Criterium	kendall tau	significantie	kendall tau	significantie	kendall tau	significantie
LOC	0,3333	0,251	0,2222	0,466	0,1111	0,754
NCLOC	0,3889	0,175	0,2778	0,348	0,0556	0,917
%COM	0,0556	0,917	-0,1667	0,602	**0,6111**	**0,029**
%Duplicaten	0,1111	0,754	0,0000	1,000	0,0000	1,000
%LOC CC>10	0,4444	0,118	0,1111	0,754	0,1111	0,754
%LOC CC>20	0,3099	0,295	-0,0282	1,000	0,2535	0,402
%LOC CC>50	0,4789	0,093	0,1409	0,675	0,1972	0,529
G(Paden/Unit)	0,3333	0,251	0,2222	0,466	0,0000	1,000
%LOC Units>10	0,1667	0,602	-0,0556	0,917	0,2778	0,348
%LOC Units>50	**0,6111**	**0,029**	0,2778	0,348	0,1667	0,602
%LOC Units>100	0,5000	0,076	0,1667	0,602	0,1667	0,602
G(LOC/Unit)	0,1667	0,602	-0,0556	0,917	0,2778	0,348

Tabel 6 Berekende correlaties voor incidenten

Uit Tabel 6 blijkt dat voor incidenten de hypothese verworpen is voor alle gemeten grootheden, met twee uitzonderingen. Dit zijn de correlatie tussen het percentage witregels en commentaarregels en de gemiddelde oplostijd van een melding en de correlatie tussen het percentage regels code van units groter dan 50 regels en het aantal bestede uren.

	uren		meldingen		uren per melding	
Criterium	kendall tau	significantie	kendall tau	significantie	kendall tau	significantie
LOC	-0,2222	0,466	-0,3099	0,295	0,0000	1,000
NCLOC	-0,1667	0,602	-0,2535	0,402	-0,0556	0,917
%COM	-0,3889	0,175	-0,3662	0,208	0,1667	0,602
%Duplicaten	-0,1111	0,754	-0,1972	0,529	0,0000	1,000
%LOC CC>10	-0,1111	0,754	-0,0845	0,834	0,0000	1,000
%LOC CC>20	-0,1409	0,675	-0,1143	0,752	0,0282	1,000
%LOC CC>50	-0,1409	0,675	-0,1143	0,752	0,0282	1,000
G(Paden/Unit)	-0,2222	0,466	-0,1972	0,529	-0,1111	0,754
%LOC Units>10	-0,5000	0,076	**-0,5916**	**0,036**	0,2778	0,348
%LOC Units>50	-0,1667	0,602	-0,1409	0,675	0,0556	0,917
%LOC Units>100	-0,0556	0,917	-0,0282	1,000	-0,0556	0,917
G(LOC/Unit)	-0,0556	0,917	-0,1409	0,675	0,0556	0,917

Tabel 7 Berekende correlaties voor reguliere wijzigingen

Uit Tabel 7 blijkt dat voor reguliere wijzigingen de hypothese verworpen is voor alle gemeten grootheden, met één uitzondering. Dit is de correlatie tussen het percentage regels code van units groter dan 10 regels en het aantal meldingen. Hiervoor is geen plausibele verklaring voorhanden, dit mag beschouwd worden als een toevallige correlatie.

5.3.2 Correlaties tussen gangbare beoordelingscriteria en beheersinspanning

	uren		meldingen		uren per melding	
Criterium	kendall tau	significantie	kendall tau	significantie	kendall tau	Significantie
Tabellen	0,4444	0,118	0,5000	0,076	0,2222	0,466
Kolommen	0,1667	0,602	0,1111	0,754	-0,0556	0,917
1-N-relaties	0,2222	0,466	0,2778	0,348	0,0000	1,000
M-N-relaties	0,0333	1,000	0,1667	0,643	-0,3667	0,246
Aantal client modules	0,5714	0,063	**0,7143**	**0,019**	0,2143	0,536
Aantal forms+reports	0,5000	0,108	**0,6429**	**0,035**	0,1429	0,711
Aantal forms	0,5455	0,081	0,5455	0,081	0,3273	0,319
Aantal libraries	0,1091	0,803	0,4001	0,212	-0,1091	0,803
Aantal reports	**0,6429**	0,035	**0,6429**	0,035	0,4286	0,174

Tabel 8 Berekende correlaties voor totaal onderhoud

Uit Tabel 8 blijkt dat voor alle gangbare beoordelingscriteria de hypothese dat er een correlatie is tussen de gemeten grootheid en het aantal bestede uren kan worden verworpen, behalve het aantal reports. Wat betreft de correlatie met het aantal meldingen kon de hypothese ook niet verworpen worden voor het totaal aantal client modules en het aantal forms plus reports.
Er zijn ook geen significante correlaties gevonden met de gemiddelde oplostijd van een melding.

	uren		meldingen		uren per melding	
Criterium	kendall tau	significantie	kendall tau	significantie	kendall tau	Significantie
Tabellen	0,4444	0,118	0,3333	0,251	0,2222	0,466
Kolommen	0,0556	0,917	0,0556	0,917	0,1667	0,602
1-N-relaties	0,2222	0,466	0,2222	0,466	0,1111	0,754
M-N-relaties	0,3000	0,353	0,3667	0,246	-0,2333	0,486
Aantal client modules	0,5714	0,063	0,2143	0,536	0,4286	0,174
Aantal forms+reports	0,5000	0,108	0,1429	0,711	0,3571	0,266
Aantal forms	0,4001	0,212	0,0364	1,000	0,4728	0,135
Aantal libraries	0,4001	0,212	0,2546	0,454	-0,0364	1,000
Aantal reports	0,3571	0,266	0,2857	0,386	0,0714	0,902

Tabel 9 Berekende correlaties voor incidenten

Uit Tabel 9 blijkt dat voor incidenten geen enkele significante correlatie gevonden is.

	uren		meldingen		uren per melding	
Criterium	kendall tau	significantie	kendall tau	significantie	kendall tau	Significantie
Tabellen	0,0000	1,000	-0,0845	0,834	-0,2222	0,466
Kolommen	-0,2778	0,348	-0,3662	0,208	-0,1667	0,602
1-N-relaties	-0,1111	0,754	-0,2535	0,402	-0,1111	0,754
M-N-relaties	0,3667	0,246	0,2704	0,415	-0,0333	1,000
Aantal client modules	0,0000	1,000	-0,1091	0,803	0,0000	1,000
Aantal forms+reports	-0,0714	0,902	-0,1818	0,618	0,0714	0,902
Aantal forms	-0,1091	0,803	-0,2593	0,451	0,0364	1,000
Aantal libraries	-0,3273	0,319	-0,2222	0,530	-0,1818	0,618
Aantal reports	0,2143	0,536	0,1091	0,803	-0,0714	0,902

Tabel 10 Berekende correlaties voor reguliere wijzigingen

Uit Tabel 10 blijkt dat voor reguliere wijzigingen ook geen enkele significante correlatie gevonden is.

5.4 Effect van codegeneratie

Het nagenoeg ontbreken van significante correlaties voor het relatieve aantal duplicaten en voor de cyclomatische complexiteit is mogelijk het gevolg van de toegepaste ontwikkeltechnieken.
Uit Tabel 11 blijkt dat alle applicaties, behalve E, zijn gemaakt door middel van code-generatie. Hiervoor is de tool Oracle Designer gebruikt. De kolom “Headstart” geeft aan of er daarbij gebruik gemaakt is van bepaalde code-templates (Headstart is een produkt van Oracle).

Applicatie	Designer	Headstart	LOC	Duplicaten (%)	LOC met CC>50 (%)
A	+	+	159.282	31,1	4,3
B	+	-	172.057	25,7	4,5
C	+	+	318.599	37,7	1,4
D	+	-	326.361	53,6	21,4
E	-	-	19.849	37,6	0,0
F	+	?	511.455	39,0	3,9
G	+	-	164.330	32,1	0,8
H	+	+	58.329	21,5	0,0
I	+	+	207.418	34,5	2,5

Tabel 11 Gebruikte tooling en enkele resultaten van statische codeanalyse

Uit de tabel blijkt dat tussen de 21,5 ent 53,6 procent van de regels code ook elders in de applicatie wordt gebruikt. Dit is in alle gevallen een hoger percentage dan de 20% die het SIG Maintainability model als ondergrens voor de laagste score (--) hanteert. Bij de bepaling van de cyclomatische complexiteit speelt het effect van gegenereerde code ook een rol. Bij applicatie D blijkt 21,4% van de

regels code voor te komen in units met een cyclomatische complexiteit van 50. De hoogste waarden voor de cyclomatische complexiteit variëren van 219 tot 975. Het merendeel van deze programma's blijkt gegenereerde code te zijn. Dit betreft bijvoorbeeld programma's die domeinvalidaties uitvoeren. Ook bij de andere applicaties blijken de units met de hoogste waarden voor de cyclomatische complexiteit gegenereerd te zijn.

6 Evaluatie en Conclusies

6.1 Foutenbeschouwing

Bij iedere meting worden fouten gemaakt. In navolging tot de chemische analyse (mijn voormalige vakgebied), kunnen deze fouten in twee categorieën worden onderverdeeld: toevallige fouten (een maat voor de precisie van de meting) en systematische fouten (een maat voor de juistheid van de gemeten waarden).

6.1.1 Precisie van de metingen

De gemeten waarden voor de statische code-eigenschappen zijn het resultaat van tellingen over de hele populatie (regels code). De waarden voor de duplicaten en de cyclomatische complexiteit zijn percentages, dus het resultaat van een deling, afgerond op één decimaal. De precisie van de resultaten wordt bepaald door de rekenkundige precisie van de gebruikte tools: de Oracle SQL-functies count en sum en de deling in Microsoft Excel. Omdat steeds de volledige populatie in beschouwing is genomen mag bij herhaling van de meting onder dezelfde omstandigheden steeds hetzelfde resultaat worden verwacht. De toevallige fout is dus bij deze metingen verwaarloosbaar.

Hetzelfde geldt voor de bepaling van de uren, aantallen meldingen en gemiddelde oplostijd (uren/melding).

6.1.2 Juistheid van de metingen

6.1.2.1 Statische code-eigenschappen

Eén applicatie (D) bevatte enkele modules met wrapped code. Deze konden niet worden geanalyseerd. Het aantal modules en de grootte ervan was klein ten opzichte van de overige modules. Dit effect is verwaarloosd.

6.1.2.1.1 Aantal regels code (LOC)

Bij de bepaling van het aantal regels code was er in de hele populatie één module waarbij de tekst geen regeleinden bevatte. Dit is geteld als één regel. Deze module bevatte slechte enkele statements, dus het effect van deze fout op de totalen is verwaarloosd.

6.1.2.1.2 Aantal regels gedupliceerde code

De bepaling van het aantal regels gedupliceerde code hangt af van de gekozen bepalingswijze, zie § 4.4.3. Voor alle applicaties is dezelfde methode gebruikt, dus ze kunnen op dit punt onderling vergeleken worden. Bij vergelijking met andere applicaties moet hiermee rekening gehouden worden.

6.1.2.1.3 Cyclomatische complexiteit

Bij de bepaling van de cyclomatische complexiteit is de between-expressie niet meegeteld. Echter, de expressie *x between A and B* kan ook worden beschouwd als *A<= x and x <= B*. Dat wil zeggen dat per between-expressie de cyclomatische complexiteit 1 eenheid te laag is berekend. Verder is de exception handler when others niet meegeteld, omdat deze in principe op elke plaats in de code kan optreden. Programmeurs kunnen voor te voorziene excepties user defined exceptions gebruiken, maar dit is niet strikt noodzakelijk. Wanneer dat niet gedaan wordt, wordt de when others exception opgeworpen. User defined exceptions zijn wel meegeteld. Gezien de grote hoeveelheid units met een hoge cyclomatische complexiteit, mogen deze twee effecten worden verwaarloosd.
Verder zijn de complexiteit die voortkomt uit table driven methods (SQL-queries en XML-stylesheets)(6) niet meegeteld. SQL-queries worden veel gebruikt in PL/SQL-code, dus deze fout kan wel een grote invloed hebben op het meetresultaat. Er is één applicatie waarin gebruik gemaakt wordt

van XML (applicatie D), maar dit betreft maar 3 modules. Aangezien deze applicatie al de hoogste waarden voor de cyclomatische complexiteit heeft, is dit effect op de analyse te verwaarlozen.

6.1.2.1.4 Unitgrootte

Bij de bepaling van de unitgrootte zijn commentaar- en witregels meegeteld. Wanneer een package een begin-end block bevat (initialisatie), wordt deze package als een aparte unit beschouwd. De witregels en de commentaarregels die bij de program units van deze package horen en die daarbuiten als commentaar zijn geplaatst (als opschrift) worden meegeteld bij de package. Als een package veel subunits bevat is deze dus veel te groot vanwege de commentaar- en witregels. Aangezien het aantal packages klein is ten opzichte van het aantal units, kan dit effect verwaarloosd worden.

6.1.2.2 Juistheid van de gegevens uit de meldingendatabase

6.1.2.2.1 Urenmeting

De juistheid van het aantal uren staat en valt met de accuratesse waarmee de leden van het onderhoudsteam de uren hebben vastgelegd. Mogelijke afwijkingen zouden kunnen zijn:
-de uren die bij een probleem of wijziging zijn geregistreerd zijn relatief laag uitgevallen om dat deze bij het incident dat eraan ten grondslag lag zijn geregistreerd (analysetijd).
-uren zijn om budgetredenen bij een ander type melding geboekt.
-indirecte uren (pauzes, vergaderingen) zijn geboekt als uren bij een melding.
Er zijn echter in het geval van dit onderzoek geen redenen bekend die aanleiding geven om te stellen dat hierdoor grote vertekeningen zijn ontstaan.

6.1.2.2.2 Aantallen meldingen

Incidenten worden voornamelijk door eindgebruikers ingediend, wijzigingen en problemen worden naar aanleiding hiervan door medewerkers van het onderhoudsteam aangemaakt. Alleen meldingen waarop uren zijn geboekt zijn meegeteld. In het geval van incidenten kan het gebeuren dat er geen bestede uren worden vastgelegd, wanneer bijvoorbeeld de oplostijd zeer kort is of wanneer de oplossing van een probleem of wijziging ook de oplossing van een incident is. Daarnaast kunnen er voor één gebeurtenis door verschillende gebruikers incidenten worden gemeld. Het tellen van incidenten is dus lastig. Voor wijzigingen en problemen speelt deze situatie nauwelijks. Bij dit onderzoek is de aanname gedaan dat voor elke unieke gebeurtenis één incident is waarop uren zijn geboekt.

6.1.2.2.3 Gemiddelde oplostijd

De gemiddelde oplostijd is het resultaat van een deling, namelijk het aantal uren gedeeld door het aantal meldingen. Dit mag echter alleen als de verdeling van de uren over de meldingen normaal verdeeld is. Dit is niet verder onderzocht, maar het is twijfelachtig of aan deze voorwaarde is voldaan. Eerder mag verwacht worden dat de uren verdeeld zijn volgens een Pareto- of een power law verdeling. In dat geval is het juister om de mediaan te bepalen. Deze ongeldige aanname kan de oorzaak zijn van het feit dat er nauwelijks correlaties gevonden zijn voor de gemiddelde oplostijd.

6.1.3 Externe factoren

In dit onderzoek is uitsluitend gekeken naar interne factoren die invloed hebben op de beheersinspanning. Er zijn echter ook een aantal externe factoren die invloed kunnen hebben op de het aantal meldingen en de benodigde oplostijd. Hiervoor zijn bij dit onderzoek aannames gedaan.

Een eerste factor is dat de oplostijd per medewerker kan verschillen, afhankelijk van zijn of haar ervaring. De omvang van het team en het klimaat binnen het team zijn echter zodanig dat uitwisseling van informatie gemakkelijk gaat. Aangenomen mag worden dat de teamleden gelijkwaardig zijn.

Een tweede factor is de stabiliteit van de beheerorganisatie. Het feit dat informatie-uitwisseling binnen het team gemakkelijk gaat betekent ook dat wisselingen in het team worden gecompenseerd weinig invloed hebben. De grootte van het team kan invloed hebben op de werkdruk, dus mogelijk op de kwaliteit van de oplossingen. Aangenomen mag worden dat dit effect in het meldingensysteem nauwelijks zichtbaar is, omdat een niet-opgeloste melding meestal heropend wordt.

Een derde factor is het beginpunt van de meting, oftewel hoeveel onderhoud er al de applicatie is gedaan door de ontwikkelorganisatie of een andere onderhoudsorganisatie. Voor de onderzochte applicaties was hierover geen informatie beschikbaar. Er is aangenomen is dat alle applicaties (na een nazorgperiode door de ontwikkelorganisatie) direct aan de beheerorganisatie zijn overgedragen. Een herhaling van dit onderzoek op een andere populatie applicaties kan aan het licht brengen of deze aanname terecht is geweest.

Een vierde factor is de stabiliteit van de applicaties (E en H waren nog maar kort in beheer). Aangenomen is dat in de eerste twee jaren de meeste meldingen worden gedaan. Daarbij komt dat het de kleinste applicaties zijn en de minst complexe applicaties, dus dat er ook relatief weinig nieuwe meldingen bij zullen komen. Deze applicaties bleken er bij het bepalen van de correlaties ook niet uit te springen.

Een vijfde factor is de informatievoorziening door de eigenaar van de applicatie. Er zijn echter geen concrete waarnemingen bekend dat dit een knelpunt is geweest bij het oplossen van meldingen.

Er zijn ook twee externe factoren die invloed hebben op het aantal meldingen, namelijk het aantal gebruikers dat de applicatie tegelijk gebruikt en de mate waarin een applicatie bedrijfskritisch is. Aangenomen is dat deze factoren eerder een rol spelen bij incidenten dan bij onderhoud.

6.2 Conclusies

6.2.1 Toepassing van het model.

Toepassing van het SIG-maintainability model op 9 applicaties toonde weinig verschillen in ISO 9126 subkarakteristieken analyseerbaarheid, wijzigbaarheid en testbaarheid. Dit komt doordat de bepaling van duplicaten en unitgrootte voor alle applicaties dezelfde scores opleverden. Voor Oracle Forms applicaties is het model niet onderscheidend genoeg.

6.2.2 Volumebepaling

De hypothese dat er een verband is tussen het aantal regels code enerzijds en anderzijds het aantal benodigde uren voor onderhoud en het aantal meldingen m.b.t. onderhoud kon niet worden verworpen.

Voor de gangbare volumebepalingen, zoals het aantal tabellen, kolommen en client modules is deze hypothese wel verworpen. Een uitzondering hierop is het aantal reports. Dit bevestigt de gangbare mening onder Oracle-programmeurs dat het onderhouden van reports lastig is.

Voor het schatten van de benodigde onderhoudsinspanning voor een applicatie is het aantal regels code dus mogelijk een bruikbare maat. Echter, niet alle applicaties voldeden aan het gevonden patroon, dus er moet rekening gehouden worden dat er andere factoren kunnen zijn die een sterkere invloed hebben op de te verwachten onderhoudsinspanning, zoals de complexiteit. Dit toont aan dat het SIG-maintainability model terecht uit meer dan één component bestaat.

De gangbare volumebepalingen zijn zoals verwacht (zie § 2.1) niet bruikbaar.

6.2.3 Complexiteit

De hypothese dat er een verband is tussen de cyclomatische complexiteit enerzijds en anderzijds het aantal benodigde uren voor onderhoud en het aantal meldingen m.b.t. onderhoud is in het algemeen verworpen.

Uitzonderingen bij dit onderzoek waren de correlatie tussen het percentage regels code met een cyclomatische complexiteit groter dan 50 en het aantal meldingen voor onderhoud en de (positieve) correlatie tussen de verdeling van de cyclomatische complexiteit over de units en het aantal meldingen voor onderhoud.

Er zijn twee factoren aan te wijzen die de bruikbaarheid van de cyclomatische complexiteit als indicator voor de benodigde onderhoudsinspanning negatief beïnvloeden, namelijk het effect van codegeneratie en het effect van table driven methods (in het geval van Oracle Forms applicaties het gebruik van SQL).
Codegeneratie levert relatief hoge waarden voor de cyclomatische complexiteit op, terwijl dat niet ongunstig hoeft te zijn voor de benodigde hoeveelheid onderhoud. Het nemen van beslissingen op basis van SQL-queries levert per query een toename van slechts 1 op van de cyclomatische complexiteit. Deze queries kunnen echter zeer complex zijn. Een methodiek om de SQL-logica mee te tellen bij de bepaling van de cyclomatische complexiteit zal verder moeten worden onderzocht.

Omdat PL/SQL een heterogene taal is, is de waarde van de cyclomatische complexiteit voor het voorspellen van de benodigde hoeveelheid onderhoud niet bruikbaar.

Bepaling van de complexiteit van het datamodel, het aantal 1-N-relaties en het aantal M-N-relaties levert ook geen significante correlaties op. Voor de het voorspellen van de benodigde onderhoudsinspanning zijn deze metrieken dus niet bruikbaar.

6.2.4 Duplicaten

De hypothese dat er een verband is tussen de het percentage gedupliceerde code enerzijds en anderzijds het aantal benodigde uren voor onderhoud en het aantal meldingen m.b.t. onderhoud is verworpen.

Dit resultaat mag worden toegeschreven aan het feit dat bij de bouw van de applicaties gebruik is gemaakt van codegeneratie. In het geval van codegeneratie gaat de aanname dat het dupliceren van code meer onderhoud vergt niet op (7). Codegeneratie wordt juist gebruikt om onderhoud eenvoudiger te maken. Dit effect blijkt niet uit dit onderzoek.

Wanneer bij applicaties gebruik is gemaakt van codegeneratie is de meting van het percentage gedupliceerde code niet bruikbaar voor de schatting van de benodigde onderhoudsinspanning.

6.2.5 Unitgrootte

De hypothese dat er een verband is tussen de het percentage regels code in units groter dan N regels en het aantal benodigde uren voor onderhoud kon niet worden verworpen voor de grenswaarde N = 50, voor de andere grenswaarden N = 10 en N = 100 is de hypothese wel verworpen.

In Oracle PL/SQL zijn units van 50 regels klein, dit verklaart dat de grenswaarde van 10 regels geen verband laat zien. Dit wijst erop dat naarmate het aandeel kleine units groter is de onderhoudsinspanning lager is. In dat geval zou echter ook een verband mogen worden verwacht tussen de Gini-coëfficiënt als maat voor de verdeling van de regels code over de units en de benodigde onderhoudsinspanning, maar dat is niet aangetoond. De bepaling van de verdeling volgens het SIG-maintainability model levert voor alle applicaties dezelfde (lage) score op.

Deze waarnemingen harmoniëren niet met elkaar, dus de bruikbaarheid van de unitgrootte als criterium voor het voorspellen van de benodigde onderhoudsinspanning zal nog verder moeten worden onderzocht. Onderzocht kan worden of andere grenswaarden betere correlaties opleveren.

6.2.6 Incidenten

De hypothese dat er een verband is tussen het aantal incidenten en de gemeten statische code-eigenschappen is in bijna alle gevallen verworpen.

Voor de uitzonderingen, de correlatie tussen het percentage regels code in units groter dan 50 regels en het aantal bestede uren en de (positieve) correlatie tussen het percentage commentaar- en witregels en de gemiddelde oplostijd, is geen duidelijke verklaring voorhanden. Deze moeten vooralsnog als toevallige correlaties worden beschouwd.

Incidenten zijn dus niet een gevolg van gebrekkige codekwaliteit. De oorzaak moet eerder gezocht worden bij architectuur- en ontwerpkeuzes. Dit valt echter buiten het blikveld van dit onderzoek.

6.3 Eindconclusie

Het SIG-maintainability-model is beperkt bruikbaar, zeker om de benodigde onderhoudsinspanning voor een bepaalde applicatie te kunnen voorspellen. Bij het gebruik moet rekening worden gehouden met de principes volgens welke de applicatie is ontwikkeld. Lage scores zeggen niet direct iets over de codekwaliteit, maar vragen om een verantwoording van de gebruikte ontwikkelmethodes. Voor analysedoeleinden is het model daarom wel geschikt. Verwacht mag worden dat wanneer een ontwikkelorganisatie bij een beheerorganisatie verantwoording geeft over de gebruikte ontwikkelmethodes, de kwaliteit van de over te dragen software hoger zal zijn - zoals in (4) is betoogd.

Een tool als Toad Code Xpert (deel van de Toad Professional Edition), aangevuld met de Sonar plugin voor het analyseren van Forms PL/SQL (http://www.sonarsource.com/plugins/plugin-plsql/), heeft dus voor het voorspellen van de onderhoudsinspanning weinig waarde. De tool is niet gemaakt voor het vergelijken van applicaties, maar voor het monitoren van de codekwaliteit tijdens de ontwikkelfase (van één applicatie), maar dat valt buiten het blikveld van dit onderzoek.

7 Bibliografie

1. **Marsman, J.H., et al.** Determination of Degree of Substitution on Extruded Benzylated Starch by H-NMR and UV Spectrometry. *Starch/Stärke.* 1990, Vol. 42, Nr. 5, pp. 191-196 (acklowledgements).
2. **Gonzalez, Reyes, Gasco, Jose en Llopis, Juan.** Information systems outsourcing: A literature analysis. *Information & management.* 2006, Vol. 43, pp. 821-834.
3. **Raylich, V. en Bennett, K.** A staged model for the software lifecycle. *Computer.* 2000, Vol. 33, Nr. 7, pp. 66-71.
4. **Zwaan, H.** Ten minste houdbaar tot - De praktijk van applicatielifecyclemanagement. *Beheer.* 2009, Nr. 6, pp. 9-13.
5. **Oppedijk, Frank R.** *Comparison of the SIG Maintainability Model and the Maintainability Index, Master Thesis Universiteit van Amsterdam.* Juli 2008.
6. **McConnell, Steve.** *Code Complete.* Washington : Microsoft Press, 2e druk 2004.
7. **Thummalapenta, Suresh, et al.** An empirical studiy on the maintenance of source clones. *Empir. Software Eng.* 2010, Vol. 15, pp. 1-34.

BIJLAGE A Antlr PL/SQL-grammatica

Deze PL/SQL-grammatica bevat geen volledige beschrijving van de taal, maar is toegesneden op het kunnen analyseren van program units (eiland parsing) met Antlr. Program units worden geconverteerd naar een xml-structuur zodat geneste units onderscheiden kunnen worden.

```
grammar Plsql;

@members {
    public StringBuffer xmlText = new StringBuffer();
    public StringBuffer getXMLText() {
      return xmlText;
    }
}

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

sourceFile options {
  k=10;
  backtrack=true;
  memoize=true;
  }     :     {xmlText = xmlText.append("<?xml version=\"1.0\" encoding='UTF-8'?>" + "\n");
                xmlText = xmlText.append("<SourceFile>" + "\n");}
                .* (programUnit)*
             {xmlText = xmlText.append("</SourceFile>");}
        ;

programUnit
        :       packageBody
        |       packageSpec
        |       procedure
        |       function
        |       trigger
        ;

declarations
        :       (IS|AS) .* ((procedureDec|functionDec|procedure|function) .*)*
        ;

caseExpr:       CASE .* (WHEN .* THEN .*)+ (ENDCASE|END)
        ;

procedureDec
        :       PROCEDURE puId parameterList? ';'
        ;
functionDec
        :       FUNCTION puId parameterList? RETURN datatype ';'
        ;

packageBody
        :       {xmlText = xmlText.append("<ProgramUnit>" + "\n");
                 xmlText = xmlText.append("<Type>Package Body</Type>" + "\n");}
                PACKAGE BODY id=puId declarations (BEGIN .+)? END ';'
                {xmlText = xmlText.append("<Name>" + $id.text + "</Name>" + "\n");

                 xmlText = xmlText.append("<Text>" + $packageBody.text + "</Text>" + "\n");
                 xmlText = xmlText.append("</ProgramUnit>" + "\n");}
        ;
packageSpec
        :       {xmlText = xmlText.append("<ProgramUnit>" + "\n");
                 xmlText = xmlText.append("<Type>Package</Type>" + "\n");}
                PACKAGE id=puId (IS|AS) .* ((procedureDec|functionDec) .*)* END ';'
                {xmlText = xmlText.append("<Name>" + $id.text + "</Name>" + "\n");
                 xmlText = xmlText.append("<Text>" + $packageSpec.text + "</Text>" + "\n");

                 xmlText = xmlText.append("</ProgramUnit>" + "\n");}
        ;
procedure
        :       {xmlText = xmlText.append("<ProgramUnit>" + "\n");
                 xmlText = xmlText.append("<Type>Procedure</Type>" + "\n");}
                PROCEDURE id=puId parameterList? declarations block
```

```
                {xmlText = xmlText.append("<Name>" + $id.text + "</Name>" + "\n");
                 xmlText = xmlText.append("<Text>" + $procedure.text + "</Text>" + "\n");
                 xmlText = xmlText.append("</ProgramUnit>" + "\n");}

        ;
function:       {xmlText = xmlText.append("<ProgramUnit>" + "\n");
                 xmlText = xmlText.append("<Type>Function</Type>" + "\n");}
                FUNCTION id=puId parameterList? RETURN datatype declarations block
                {xmlText = xmlText.append("<Name>" + $id.text + "</Name>" + "\n");
                 xmlText = xmlText.append("<Text>" + $function.text + "</Text>" + "\n");
                 xmlText = xmlText.append("</ProgramUnit>" + "\n");}

        ;
trigger
        :       {xmlText = xmlText.append("<ProgramUnit>" + "\n");
                 xmlText = xmlText.append("<Type>Trigger</Type>" + "\n");}
                TRIGGER id=triggerId (BEFORE|AFTER|INSTEAD) .* ON .* ((procedure|function) .*)*
block
                {xmlText = xmlText.append("<Name>" + $id.text + "</Name>" + "\n");
                 xmlText = xmlText.append("<Text>" + $trigger.text + "</Text>" + "\n");
                 xmlText = xmlText.append("</ProgramUnit>" + "\n");}
        ;

parameterList
        :       '(' parameterName IN? (OUT NOCOPY?)? datatype ((DEFAULT|ASSIGN) value)? (','
parameterName IN? (OUT NOCOPY?)? datatype ((DEFAULT|ASSIGN) value)?)* ')'
        ;
datatype:       ID ('%' TYPE)
        |       ID ('%' ROWTYPE)
        |       ID
        ;
textValue
        :       QUOTED_STRING+
        |       CHAR
        |       QUOTED_STRING+ ('||' (QUOTED_STRING+|CHAR|ID))+
        |       CHAR ('||' (QUOTED_STRING+|CHAR|ID))+
        |       ID ('||' (QUOTED_STRING+|CHAR|ID))+
        ;
argument:       ID | NUMBER | textValue
        ;
functionCall
        :       ID '(' (argument (',' argument)*)? ')' ('||' textValue)?
        |       textValue '||' (ID '(' (argument (',' argument)*)? ')')?
        |       ID '(' ID '(' (argument (',' argument)*)? ')' (',' argument)* ')'
        ;
value   :       ID | NUMBER | textValue | functionCall
        ;
parameterName
        :       ID
        |       PACKAGE
        |       BODY
        |       FUNCTION
        |       TRIGGER
        |       BEFORE
        |       AFTER
        |       INSTEAD
        |       OUT
        |       NOCOPY
        |       ROWTYPE
        |       TYPE
        ;

puId    :       ID
        |       '"'ID'"'
        ;
triggerId
        :       ID
        |       '"'ID'"'('.' (('"'ID'"')|ID))*
        ;

PACKAGE :       P A C K A G E ;
BODY    :       B O D Y ;
PROCEDURE
        :       P R O C E D U R E ;
FUNCTION:       F U N C T I O N ;
TRIGGER :       T R I G G E R ;
BEFORE  :       B E F O R E ;
```

```
AFTER  :      A F T E R ;
INSTEAD:      I N S T E A D ;
ON     :      O N ;
IS     :      I S ;
AS     :      A S ;
RETURN :      R E T U R N ;
BEGIN  :      B E G I N ;
CASE   :      C A S E ;
WHEN   :      W H E N ;
THEN   :      T H E N ;
ENDIF  :      E N D (' ')+ I F WS*;
ENDLOOP:      E N D (' ')+ L O O P ((' ')+ ID)? WS*;
ENDCASE:      E N D (' ')+ C A S E WS*;
END    :      E N D ((' ')+ ID)? WS*;IN    :      I N ;
OUT    :      O U T ;
NOCOPY :      N O C O P Y ;
DEFAULT:      D E F A U L T ;
ROWTYPE:      R O W T Y P E ;
TYPE   :      T Y P E ;
CHAR   :      C H R '(' NUMBER ')' ;

block
       :      BEGIN .* (caseExpr|block .*)* END ';'
       ;


/*------------------------------------------------------------------
 * LEXER RULES
 *------------------------------------------------------------------*/
ML_COMMENT
       :      '/*' (options {greedy=false;} : .)* '*/' {$channel=HIDDEN;}
       ;
EL_COMMENT
       :      '--' (options {greedy=false;} : .)* '\n' {$channel=HIDDEN;}
       ;
QUOTED_STRING
       :      '\'' (options {greedy=false;} : .)* '\''
       ;

WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        ) {$channel=HIDDEN;}
    ;

ID  :  ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_'|'$'|'#'|'.'|'-')*
    ;
NUMBER :      ('-'|'+')? ('0'..'9') ('0'..'9'|'.')*
       ;
ASSIGN :      ':='
       ;

ANY    :      .
       ;

fragment A:('a'|'A');
fragment B:('b'|'B');
fragment C:('c'|'C');
fragment D:('d'|'D');
fragment E:('e'|'E');
fragment F:('f'|'F');
fragment G:('g'|'G');
fragment H:('h'|'H');
fragment I:('i'|'I');
fragment J:('j'|'J');
fragment K:('k'|'K');
fragment L:('l'|'L');
fragment M:('m'|'M');
fragment N:('n'|'N');
fragment O:('o'|'O');
fragment P:('p'|'P');
fragment Q:('q'|'Q');
fragment R:('r'|'R');
fragment S:('s'|'S');
fragment T:('t'|'T');
fragment U:('u'|'U');
```

```
fragment V:('v'|'V');
fragment W:('w'|'W');
fragment X:('x'|'X');
fragment Y:('y'|'Y');
fragment Z:('z'|'Z');
```

BIJLAGE B Meetgegevens

Applicatie	LOC	NCLOC	Modules	Units	LOC CC>0	LOC CC>10	LOC CC>20	LOC CC>50	LOC Dup>=6
A	42.918	20.216	134	562	38.030	14.086	11.708	5.190	4.393
B	62.473	30.718	139	698	59.609	36.765	28.035	6.180	12.248
C	191.508	108.354	798	4.192	163.977	35.147	17.214	4.374	62.719
D	326.361	260.732	280	4.560	287.189	99.503	84.365	66.997	174.848
E	1.898	1.430	32	36	1.808	262	0	0	264
F	307.364	169.227	230	5.606	258.716	71.941	42.484	17.898	94.721
G	56.632	39.606	247	1.121	46.255	11.054	4.914	797	11.558
H	2.888	2.066	26	63	2.281	753	122	0	238
I	90.539	60.047	543	1.651	75.308	23.177	12.052	4.556	29.354

Tabel 12 Meetresultaten database software

Applicatie	LOC	NCLOC	Modules	Units	LOC CC>0	LOC CC>10	LOC CC>20	LOC CC>50	LOC Dup>=6
A	116.364	83.785	78	4.381	99.013	20.819	9.441	1.071	45.162
B	109.584	78.589	168	4.677	105.491	18.951	6.528	1.488	32.016
C	127.091	102.879	103	4.192	119.248	21.187	3.416	0	57.240
D	-	-	-	-	-	-	-	-	-
E	17.951	12.430	48	1.230	17.903	3.440	1.776	0	7.193
F	204.091	159.927	147	9.730	195.002	36.856	10.356	1.751	104.861
G	107.698	74.648	145	5.293	94.543	17.813	5.007	363	41.191
H	55.441	39.650	46	2.214	47.210	10.681	2.772	0	12.318
I	116.879	90.410	96	5.535	107.288	20.299	4.656	522	42.298

Tabel 13 Meetresultaten client software

Applicatie	LOC	NCLOC	Modules	Units	LOC CC>0	LOC CC>10	LOC CC>20	LOC CC>50	LOC Dup>=6
A	159.282	104.001	212	4.943	137.043	34.905	21.149	6.261	49.555
B	172.057	109.307	307	5.375	165.100	55.716	34.563	7.668	44.264
C	318.599	211.233	901	8.384	283.225	56.334	20.630	4.374	119.959
D	326.361	260.732	280	4.560	287.189	99.503	84.365	66.997	174.848
E	19.849	13.860	80	1.266	19.711	3.702	1.776	0	7.457
F	511.455	329.154	377	15.336	453.718	108.797	52.840	19.649	199.582
G	164.330	114.254	392	6.414	140.798	28.867	9.921	1.160	52.749
H	58.329	41.716	72	2.277	49.491	11.434	2.894	0	12.556
I	207.418	150.457	639	7.186	182.596	43.476	16.708	5.078	71.652

Tabel 14 Meetresultaten database- en client software gesommeerd

Applicatie	LOC UNITS>0	LOC UNITS>10	LOC UNITS>50	LOC UNITS>100
A	36.590	36.136	28.291	20.389
B	57.685	56.821	49.331	42.913
C	118.671	101.989	35.515	7.863
D	267.014	260.967	205.509	160.014
E	1.808	1.785	1.310	705
F	243.593	237.685	158.405	95.549
G	92.496	76.885	31.135	12.326
H	45.227	39.922	17.723	5.161
I	105.297	90.659	38.304	10.474

Tabel 15 Meetresultaten database software (vervolg)

Applicatie	LOC UNITS>0	LOC UNITS>10	LOC UNITS>50	LOC UNITS>100
A	98.409	88.127	41.386	16.245
B	103.791	90.905	39.237	17.913
C	153.877	149.240	81.422	45.344
D	-	-	-	-
E	17.903	15.602	5.291	2.441
F	194.560	167.268	67.962	16.673
G	42.758	41.291	27.902	18.197
H	2.281	2.237	1.613	927
I	74.756	72.771	49.745	33.197

Tabel 16 Meetresultaten client software (vervolg)

Applicatie	LOC UNITS>0	LOC UNITS>10	LOC UNITS>50	LOC UNITS>100
A	134.999	124.263	69.677	36.634
B	161.476	147.726	88.568	60.826
C	272.548	251.229	116.937	53.207
D	267.014	260.967	205.509	160.014
E	19.711	17.387	6.601	3.146
F	438.153	404.953	226.367	112.222
G	135.254	118.176	59.037	30.523
H	47.508	42.159	19.336	6.088
I	180.053	163.430	88.049	43.671

Tabel 17 Meetresultaten database- en client software gesommeerd (vervolg)

Applicatie	G (LOC/Module)	G (LOC/Unit)	G (CC/Unit)
A	0,7867	0,6441	0,5373
B	0,6794	0,6911	0,5186
C	0,7827	0,6865	0,5092
D	0,6171	0,8187	0,6099
E	0,6393	0,7204	0,4831
F	0,4844	0,8877	0,5357
G	0,7438	0,6718	0,5058
H	0,7947	0,5635	0,5001
I	0,7777	0,6531	0,5333

Tabel 18 Berekende Gini-coëfficienten

Applicatie	Tabellen	Kolommen	1-N relaties	M-N relaties	Views
A	74	2.868	55	0	130
B	129	1.278	22	0	16
C	126	3.176	204	5	156
D	66	1.471	53	2	44
E	41	630	4	0	1
F	221	3.938	137	0	77
G	174	2.768	111	5	173
H	35	487	17	0	22
I	127	2.843	75	1	103

Tabel 19 Klassieke applicatiekarakteristieken - database

Applicatie	Forms	Libraries	Reports	Forms+Reports	Client modules
A	43	35	0	43	78
B	103	15	50	153	168
C	74	16	13	87	103
D	0	0	0	0	0
E	28	1	19	47	48
F	74	21	52	126	147
G	59	42	44	103	145
H	24	15	7	31	46
I	60	14	22	82	96

Tabel 20 Klassieke applicatiekarakteristieken - client modules

BIJLAGE C Criteria voor bepaling SIG ratings

Deze bijlage is ontleend aan: http://docs.codehaus.org/display/SONAR/SIG+Maintainability+Model

Volume: based on the number of lines of code

Rank	LOC
--	> 1310000
-	> 655000
0	> 246000
+	> 66000
++	> 0

Duplications: based on the density of duplications

Rank	Duplication
--	> 20%
-	> 10%
0	> 5%
+	> 3%
++	> 0%

Complexity: based on the cyclomatic complexity of methods
A first pass consists in determining the percentage of LOC belonging to methods that are in a certain range of complexity

Eval	Complexity
Very high	> 50
High	> 20
Medium	> 10
Low	> 0

Then based on the distribution, we use the following table to evaluate the rank :

Rank	Medium	High	Very High
++	< 25%	< 0%	< 0%
+	< 30%	< 5%	< 0%
0	< 40%	< 10%	< 0%
-	< 50%	< 15%	< 5%

Rank is -- otherwise

Unit size: based on the lines of code of methods
A first pass consists in determining the percentage of LOC belonging to methods that are in a certain range of LOCS

Eval	LOCs
Very high	> 100
High	> 50
Medium	> 10
Low	> 0

Then based on the distribution, we use the following table to evaluate the rank :

Rank	Medium	High	Very High
++	< 25%	< 0%	< 0%
+	< 30%	< 5%	< 0%
0	< 40%	< 10%	< 0%
-	< 50%	< 15%	< 5%

Rank is -- otherwise