Gebruik van de Wijzigingshistorie voor het aantonen van Logical Couplings binnen een SOA

Aldert C. Boerhoop

Aldert C. Boerhoop
30/8/2007

Master Software Engineering

Afstudeerdocent :	Jan van Eijck
Stagebegeleider:	Bas Meerman, Gert Jan Timmerman
Opdrachtgever:	Bas Meerman
Publicatiestatus:	Openbaar

Universiteit van Amsterdam

Universiteit van Amsterdam

InfoSupport

I. | Voorwoord

Ik heb mijn afstudeertijd bij Info Support ervaren als uiterst leerzaam en daarnaast ook erg gezellig. Binnen het bedrijf heerst een zeer sterke saamhorigheid waardoor de sfeer erg goed is. Daarnaast staat kennis hoog in het vaandel en dat wordt binnen de organisatie dan ook duidelijk uitgedragen.

Op de eerste plaats wil ik graag de begeleiding binnen Info Support bedanken voor hun inzet om de opdracht tot een succes te maken. Bas Meerman mijn opdrachtgever, stroomde over van de ideeën en wist me daardoor altijd genoeg input te geven zodat ik mijn opdrachtomschrijving verder kon verfijnen. Dennis Joosten, Manager Professional Development Center, bedank ik voor de ideeën betreffende het Proof of Concept. Ook de verschillende werknemers van Info Support die een bijdrage hebben geleverd aan het onderzoek wil ik bedanken. Mijn technische begeleider Gert-Jan Timmerman wil ik graag bedanken voor de zeer kritische feedback op de verschillende documenten.

Daarnaast wil ik mijn begeleider aan de UVA Jan van Eijck bedanken voor de begeleiding tijdens het onderzoek en de feedback op de verschillende documenten. Ook Jurgen Vinju wil ik bedanken voor zijn inzet en begeleiding.

Als laatste dank ik mijn ouders voor de mogelijkheid die ze mij gegeven hebben om een masterstudie te volgen en voor de steun tijdens het onderzoek. Ook mijn vriendin wil ik bedanken voor haar steun.

Het resultaat van mijn masteronderzoek ligt hier voor u, ik heb er met plezier aangewerkt. Hopelijk wordt dit ook ervaren bij het lezen ervan. Veel plezier met doorlezen.

Aldert

II. | Samenvatting

Bij Info Support worden administratieve enterprise systemen ontwikkeld met behulp van een speciaal ontwikkelde ontwikkelstraat genaamd Endeavour. Door gebruik te maken van deze ontwikkelstraat kan er op een productieve en effectieve manier software ontwikkeld worden. Binnen Endeavour wordt een Service Oriented Architecture (SOA) voorgeschreven voor de te ontwikkelen systemen. Binnen een SOA worden verschillende verantwoordelijkheden vertegenwoordigd door aparte services. Binnen Endeavour worden in de Logische Referentie Architectuur bepaalde regels opgelegd aan de verschillende services. Zo zijn er verschillende service typen die op een bepaalde manier met elkaar mogen communiceren. Het doel van het toepassen van een dergelijke architectuur is het isoleren van functionaliteit zodat onderhoud en uitbreiding op een zon goedkoop, makkelijk en consistent mogelijke manier kunnen worden toegepast.

Het probleem bij de systemen die ontwikkeld worden binnen Endeavour is om aan te tonen dat de architectuurprincipes daadwerkelijk worden toegepast en dat ze werken. Om te monitoren of de gestelde regels worden nageleefd en effect hebben, stelt dit onderzoek een methode voor. Met deze methode kunnen Logical Couplings, aan de hand van Change Dependencies, tussen de verschillende services van de SOA aangetoond en inzichtelijk gemaakt worden.

Om de methode te toetsen is er een Proof of Concept (PoC) ontwikkeld genaamd Logical Coupling Analyser (LCA). Aan de hand van dit PoC zijn verschillende testen doorlopen en is er een Reallife case Vektis onderzocht. Voor de verschillende testen zijn diverse scenario's opgesteld die de mogelijkheden van de LCA toetsen.

Uit de resultaten van de analyse blijkt dat de methode een meerwaarde levert op de bestaande kwaliteitstooling binnen de ontwikkelstraat. Binnen de onderzochte Reallife case Vektis worden verschillende Logical Couplings aangetoond. Er blijkt echter nog meer aangetoond te kunnen worden met de LCA.

Inhoud

1. | Inleiding

Tijdens de ontwikkeling van bedrijfskritische software systemen, die verschillende bedrijfsprocessen vertegenwoordigen, is kwaliteitscontrole een onderdeel dat niet mag ontbreken. Onder Kwaliteitscontrole verstaan we:

- Unit Testing
- Performance analyse
- Naming Conventions analyse
- Security analyse
- Localization analyse
- Library Design analyse

Kwaliteitscontrole maakt het mogelijk om tijdens de ontwikkeling in te grijpen wanneer het beoogde resultaat in gevaar komt. Om de continuïteit van een systeem te waarborgen is de onderhoudbaarheid een belangrijk element. De architectuur van een systeem is mede bepalend voor de onderhoudbaarheid. Binnen een architectuur is maximale isolatie gewenst zodat wijzigingen en change requests eenvoudig, snel en op een consistente manier door te voeren zijn. Binnen Info Support ontstond de vraag hoe je controles kunt uitvoeren op de status van de geïmplementeerde architectuur. In dit master onderzoek stellen we een methode voor die gebruikt kan worden als indicator om uitspraken te kunnen doen over de status van de geïmplementeerde architectuur.

Om een beter beeld te krijgen van het probleem domein zal in hoofdstuk ***2| Endeavour als Framework voor Architectuurprincipes*** ingegaan worden op de achtergrond en context van het probleem. In hoofdstuk ***3| Probleemstelling & Motivatie & Inperking*** wordt het probleem dat onderzocht is beschreven. Vervolgens wordt in hoofdstuk ***4| LCA – Een Tool voor het analyseren van Logical Couplings*** de gekozen onderzoeksmethode besproken. In hoofdstuk ***5| Toepassing van LCA op Vektis*** wordt het uiteindelijke onderzoek dat is uitgevoerd beschreven. In de laatste twee hoofdstukken ***6| Resultaten*** en ***7| Evaluatie & Conclusies & Aanbevelingen*** worden de resultaten en de bevindingen besproken. Ook worden hier mogelijke toekomstige richtingen voor het onderzoek voorgesteld.

2. | Endeavour als Framework voor Architectuurprincipes

In dit hoofdstuk wordt de achtergrondinformatie behandeld die nodig is om het probleem beschreven in hoofdstuk ***3| Probleemstelling & Motivatie & Inperking*** te begrijpen en een voorstelling te kunnen maken van het probleemdomein.

2.1 | Info Support & Endeavour

2.1.1 | Algemeen

Het onderzoek is verricht bij Info Support[1] te Veenendaal in opdracht van het Professional Development Center (PDC). De opdracht heeft betrekking op de, door Info Support ontwikkelde, ontwikkelstraat Endeavour. Info Support bestaat sinds 1986 en heeft sindsdien een gezonde financiële positie. Deze positie wordt gebruikt om continu te investeren in de kennis van de werknemers. Info Support is voornamelijk werkzaam in Nederland (250 medewerkers) en België (50 medewerkers). Het dienstenpakket dat aangeboden wordt, bestaat uit:

- Training
- Software ontwikkeling
- Beheer en ondersteuning
- Hosting

2.1.2 | Professional Development Center (PDC)

Het Professional Development Center (PDC) is een aparte afdeling binnen het bedrijf waar de ontwikkelstraat Endeavour wordt onderhouden en door ontwikkeld. Daarnaast biedt het PDC ondersteuning bij de verschillende projecten waar de ontwikkelstraat wordt ingezet. De afdeling telt 13 personen en bestaat sinds 2003. De nieuwste technologieën worden op het PDC onderzocht op inzetbaarheid en integreerbaarheid in Endeavour. Het PDC pioniert op technologisch vlak en ligt dan ook qua technologie twee jaar voor op de rest van de markt. Hierdoor kan Info Support tijdens de uitvoering van projecten met de ingezette technologie een jaar voorliggen op de rest van de markt.

2.1.3 | Endeavour

Endeavour is een Software Factory (***Definitie 2-1***) waarin de productiviteit en kwaliteit omtrent de ontwikkeling van een software systeem verbeterd worden. Dit wordt mogelijk gemaakt door verschillende informatie bronnen te combineren tot een grote portal waar een ontwikkelteam terecht kan voor de actuele werkverdeling en bijbehorende support.

[1] Voor meer informatie over Info Support: http://www.infosupport.com

Een Software Factory maakt het mogelijk om arbeidsintensieve software ontwikkeling te veranderen in kapitaalintensieve software ontwikkeling waar substantiële investeringen in gedaan kunnen worden op een acceptabel risico niveau(Fernström, et al., 1992).

Definitie 2-1, Software Factory

De ontwikkelstraat biedt naast het op RUP [2]gebaseerde ontwikkelproces ondersteuning op de daadwerkelijke implementatie van de gestelde eisen van de klant. Dit wordt gedaan aan de hand van geïntegreerde tools en status-monitors binnen Endeavour die voortdurend de kwaliteit en status van het project bewaken. Voorbeelden van dergelijke tools / status-monitors zijn:

- FXCop; analyseert de code om verschillende fouten te vinden in de volgende onderdelen:
    - Library Design
    - Localization
    - Naming conventions
    - Security
- Unit testing; wordt gebruikt om de code te testen op fouten in de functionaliteit.
- Daily build; bij Endeavour zijn continue de status van de daily build en de build errors bekend
- Code Churn; per service is bekend aan hoeveel activiteit deze is blootgesteld

Daarnaast zijn er kant en klare bouwstenen (Building Blocks) die gebruikt kunnen worden bij de ontwikkeling van een systeem. Endeavour is beschikbaar in twee smaken: .Net en Java.

2.1.4 | Service Oriented Architecture

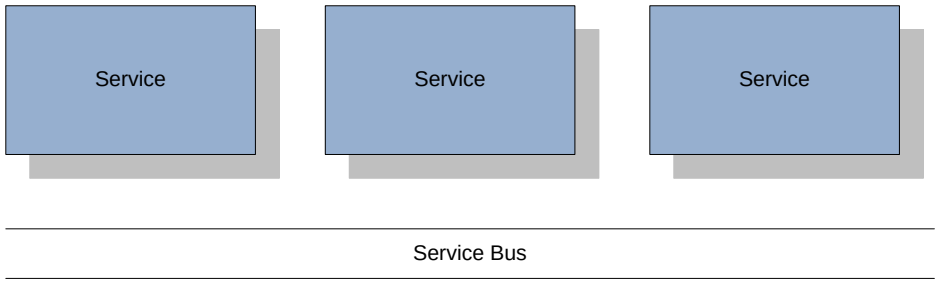
Voor het begrip Service Oriented Architectuur zijn verschillende definities te vinden in de literatuur. Tijdens dit onderzoek zal de volgende definitie gehanteerd worden:

Een Service Oriented Architecture is een heterogeen netwerk van autonome services die samen één systeem vormen en die met elkaar communiceren via een Service bus.

Definitie 2-2, SOA

Een dergelijke architectuurontwerp is geïllustreerd in ***Figuur 2-1***. Door het ontwerp is het mogelijk om verschillende systemen aan elkaar te koppelen die gebruik maken van verschillende technologieën en die geschreven zijn in verschillende talen doordat de Services elkaar niet kennen en berichten uitwisselen via de Service bus.

[2] RUP is een iteratief softwareontwikkelproces waarin verschillende principes en best practices zijn verwerkt. Daarnaast is de ontwikkeling verdeeld in verschillende fasen en zijn er verschillende diciplines die aan bod komen tijdens de ontwikkeling.



Figuur 2-1, Voorbeeld van een SOA implementatie

Een SOA-architectuur heeft verschillende karakteristieke eigenschappen (Huhns & Singh, 2005). De verschillende eigenschappen zijn hieronder (*Tabel 2-1*) toegelicht.

Eigenschap:	Toelichting:
Loose Coupling	Binnen een SOA wordt Loose Coupling toegepast om zo veel mogelijk onafhankelijke services te introduceren die alleen high-level contractuele relaties met elkaar aangaan.
Implementatie neutraal	Voor de verschillende services geldt dat de interface van belang is om communicatie mogelijk te maken. De implementatie daarentegen kan uniek zijn en zelfs in een andere taal geschreven zijn.
Flexibel in configuratie	Als gevolg van de Loose Coupling en de neutrale implementatie zullen er configuratiemogelijkheden aanwezig moeten zijn om de verschillende componenten aan elkaar te binden.
Persistent	Een service dient lang genoeg te bestaan om als een stabiele component te kunnen functioneren in de vaak dynamische heterogene omgeving waarin elke service een autonome positie inneemt. Doordat de service geen kennis heeft van andere services, maar wel deelneemt aan processen die verschillende services beslaan, moet een service lang genoeg bestaan om bijvoorbeeld corrigerende acties uit te voeren naar aanleiding van een exception.
Granulariteit	De verschillende services dienen op een hoog abstractieniveau ontworpen en begrepen te worden. Door uit te vinden en er rekening mee te houden wat de belangrijkste kwaliteiten van een bepaalde service zijn, voor de verschillende deelnemers binnen een SOA, worden het aantal berichten tussen de services verkleind en zij van meer betekenis.
Team	In plaats van een programma dat een geheel proces uitvoert, werken de verschillende services in een team om bepaalde processen te realiseren. Elke service heeft een bepaalde verantwoordelijkheid binnen de verschillende processen.

Tabel 2-1, Karakteristieke eigenschappen SOA architectuur

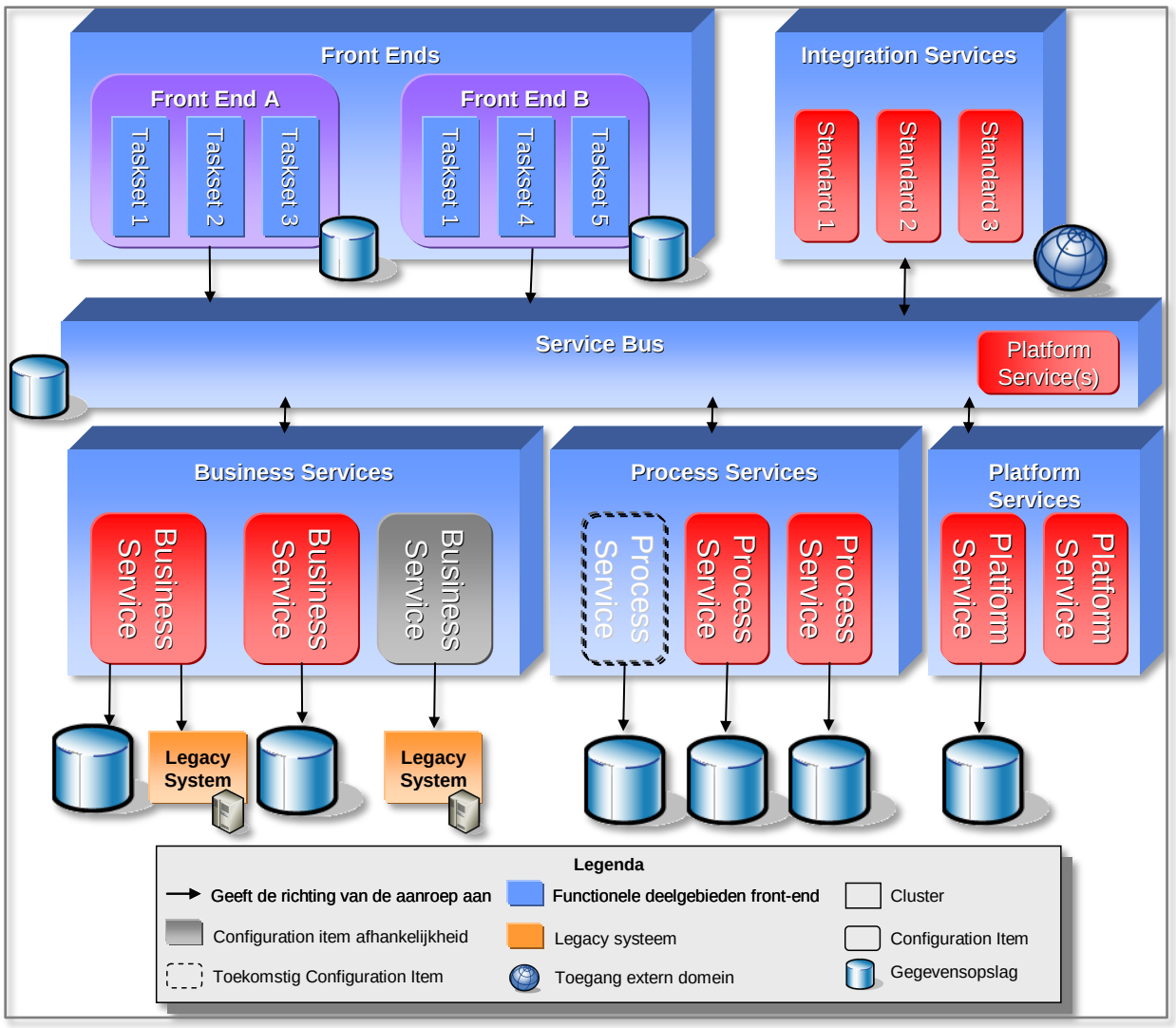
Er zijn verschillende voordelen die door het toepassen van een SOA benut kunnen worden. De verschillende voordelen worden hieronder(*Tabel 2-2*) toegelicht(Smit, 2003)(Huhns & Singh, 2005).

Voordeel:	Omschrijving:
Hergebruik	Hergebruik van bestaande systemen wordt mogelijk door deze als een service te integreren
Beschikbaarheid	De verschillende services kunnen beter beschikbaar worden gesteld voor andere systemen
Legacy integratie	Mogelijkheid tot uitbreiden van legacy systemen door deze te integreren in bestaande applicaties
Efficiëntie	Het is efficiënter om software te ontwikkelen door deze opzet

Tabel 2-2, Voordelen van het toepassen van een SOA

2.1.5 | Logische Referentie Architectuur

Binnen Endeavour wordt een architectuur voorgeschreven die gebruikt dient te worden bij de ontwikkeling van een administratieve enterprise applicatie. Deze architectuur is een SOA en is beschreven in de Logische Referentie Architectuur(LRA)(de Vries, ENDEV01 - Logical Reference Architecture, 2006). In *Figuur 2-2* is geïllustreerd hoe een architectuur binnen Endeavour eruit ziet.



Figuur 2-2, Logische Referentie Architectuur (LRA) Endeavour(de Vries, ENDEV01 - Logical Reference Architecture, 2006)

Naast de Service Bus zijn er services te onderscheiden die verschillende verantwoordelijkheden binnen het gehele systeem hebben. Binnen Endeavour wordt een service ook wel een Configuration Item (CI) genoemd. De services worden hieronder toegelicht:

Front ends

Een Front End is een onderdeel van de interface van het systeem naar de eindgebruiker. Via een Front End kan de eindgebruiker bepaalde taken uitvoeren op het systeem. Een Front End is een combinatie van een medium en een gebruikersgroep. Een Front End mag alleen services aanroepen van een ander type. Daarnaast mag hij alleen gegevens beheren die tijdelijk zijn en die van toepassing zijn op de eigen service.(de Vries, 2006)

Voorbeeld; *Een Frond End is bijvoorbeeld een client applicatie op een gebruiker zijn computer geïnstalleerd of een website waar een gebruiker naar toe kan surfen.*

Business service

Een Business Service heeft als doel gegevens te beheren en dient herkenbaar te zijn voor de gebruikersorganisatie. Een Business Service onderneemt geen actie op andere services.(de Vries, 2006)

Voorbeeld; *Een Business Service kan bijvoorbeeld persoonsgegevens beheren of bestelling gegevens.*

Process service

Een Process Service maakt het mogelijk om verschillende bedrijfsprocessen te automatiseren. Process Services kunnen gegevens beheren die alleen van kracht zijn tijdens het uit te voeren proces. Een Process Service mag met uitzondering van de Front Ends alle service typen aanroepen.(de Vries, 2006)

Voorbeeld; *Een voorbeeld van een Process service is een service die het proces verzorgt voor het aanvragen van een uitkering. In dit geval zal er input verkregen worden via de front ends en verschillende business services via de process service aangestuurd worden om het proces mogelijk te maken.*

Platform service

Een Platform Service maakt het mogelijk om bedrijfsfunctionaliteit te ontwikkelen binnen het systeem. Een Platform Service kan alleen andere Platform Services aanroepen.(de Vries, 2006)

Voorbeeld; *Een voorbeeld van een Platform service is een service die gebruikt kan worden om gebruikers te authenticeren.*

Integration service

Een Integration Service maakt het mogelijk om een extern domein aan te sluiten op het systeem. Hierdoor is het mogelijk om systemen van een ander type dan die binnen Endeavour deel uit te laten maken van het systeem. Een Integration Service heeft de mogelijkheid om bepaalde gegevens te cachen en te transformeren zodat deze geschikt zijn voor het systeem. Een Integration Service mag met uitzondering van de Front Ends alle service typen aanroepen.(de Vries, 2006)

Voorbeeld; *Een voorbeeld van een Integration service is een service die het mogelijk maakt om een extern systeem dat postcode checks mogelijk maakt te integreren binnen het systeem in ontwikkeling.*

2.2 | Code Churn

Wanneer een software systeem evolueert treden er wijzigingen op in de onderliggende source code. Naast wijzigingen worden er ook bepaalde stukken verwijderd of toegevoegd. Het is moeilijk om te begrijpen, meten en te voorspellen wat het effect is van de verschillende wijzigingen (Ball & Nagappan, 2005). Dergelijke wijzigingen in de source code vallen onder een noemer: 'Code Churn'. Door de Code Churn van een systeem in kaart te brengen kan men analyseren hoe bepaalde problemen ontstaan (Snijder, 2007).

Code Churn kan gedefinieerd worden als:

Code churn is defined as lines added, modified or deleted to a file from one version to another.

Definitie 2-3, Code Churn (AskBurton, 2004)

Code Churn is relatief eenvoudig aan te tonen; men hoeft namelijk alleen de vergelijking te maken tussen twee verschillende versies van een bepaald tekstueel gegeven de, zogenaamde file diff. Bij ***Code voorbeeld 2-1*** wordt weergegeven (***dik, cursief***) welke wijzigingen hebben plaats gevonden tussen versie 1.0 en 2.0.

Versie 1.0:

```
class ClassA
{
        public ClassA (){}

        public int returnInt()
        {
                return 8;
        }
}
```

Versie 2.0:

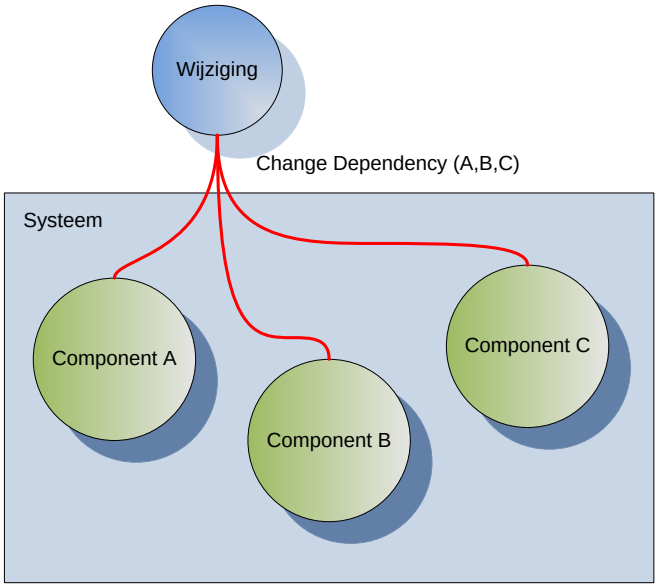
```
class ClassA
{
        private int temp;

        public ClassA ()
        {
                temp = 8;
        }

        public int returnInt()
        {
                return temp;
        }
}
```

Code voorbeeld 2-1, Code Churn (File diff)

2.3 | Change Dependencies & Logical Couplings

In software zijn er op syntactisch niveau koppelingen aan te tonen tussen de componenten van een systeem. Deze koppelingen bestaan zowel uit directe aanroepen als dynamische aanroepen tussen verschillende componenten. Maar sommige afhankelijkheden zijn niet af te leiden uit source code omdat ze niet in een afleidbare vorm zijn genoteerd. Door naar de Code Churn te kijken, die plaats vindt om een wijziging door te voeren, kunnen Change Dependencies worden aangetoond (***Figuur 2-3***). Een

Change Dependency is een afhankelijkheid op basis van een gemeenschappelijke wijziging met één en hetzelfde doel.



Figuur 2-3, Voorstelling Change Dependency

Zoals in ***Figuur 2-3*** wordt geïllustreerd: wanneer voor een bepaalde wijziging component a, b en c gewijzigd zijn, bestaat hier een Change Dependency tussen de componenten.

Voorbeeld 2-1:

Sinds 1951 worden de volgende kentekenindelingen gehanteerd:

Jaar:	Kenteken voorbeeld:
1951	XX-99-99
1965	99-99-XX
1973	99-XX-99
1978	XX-99-XX
1991	XX-XX-99
1999	99-XX-XX

Tabel 2-3, Kentekenindelingen; X = letter, 9 = cijfer

In een applicatie waarin auto's aan persoongegevens gekoppeld worden, zal een business rule afdwingen dat het kenteken van de auto voldoet aan de gestelde indelingen. De autogegevens worden beheerd door een Business service en de interface voor de gebruiker, van de applicatie, zal bestaan uit een Front end. Omdat beide services autonoom functioneren, zullen beide services de business rule afdwingen. De Front end zal testen bij het invoeren of het kenteken voldoet terwijl de business service dit zal doen wanneer deze de gegevens aangereikt krijgt om op te slaan.

Nu treedt er een wijziging op in de business rule naar aanleiding van nieuwe kentekenindelingen:

Sinds 1 september 2005 zijn de volgende kentekenindelingen toegevoegd doordat de capaciteit van de oude indelingen niet meer toereikend waren.

Indeling:	Kenteken voorbeeld:
Kentekenindeling 7	12-ABC-3
Kentekenindeling 8	1-ABC-23
Kentekenindeling 9	AB-123-C
Kentekenindeling 10	A-123-BC

Tabel 2-4, Nieuwe kentekenindelingen; A,B,C = letter, 1,2,3 = cijfer

De nieuwe kentekenindelingen hebben als gevolg dat de business rule aangepast dient te worden. Het aanpassen van de business rule veroorzaakt een wijziging zowel in de Front end als in de Business service. Wanneer de wijziging wordt doorgevoerd zal er een Change Dependency tussen de verschillende services ontstaan.

In de literatuur wordt een Change Dependency vaak gelijk gesteld aan een Logical Coupling. Wanneer we kijken naar de volgende definities zien we dat Logical Couplings als volgt worden beschreven:

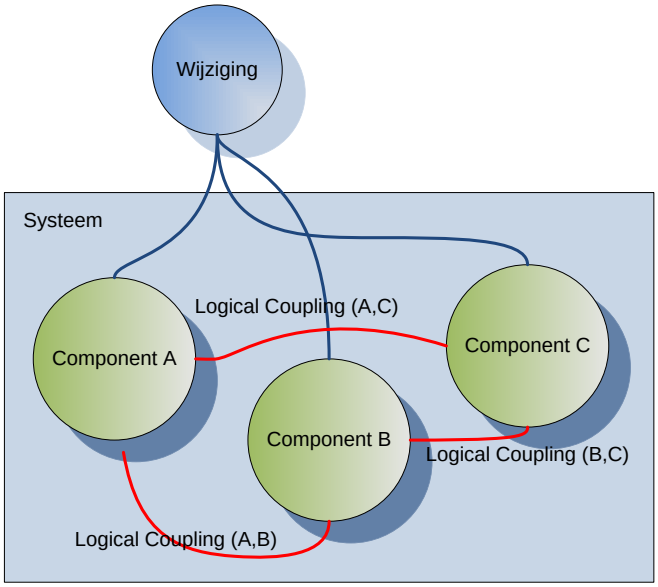
Herkenbare combinaties van componenten, die voortkomen uit Change Dependencies, worden Logical Couplings genoemd.

Definitie 2-4, Logical Coupling (Gall, et al., 2003).

Logical coupling represents the implicit dependency relationship between two or more software artifacts that have been observed to frequently change together during the evolution of the system. This co-change information can either be present in the versioning system, or must be inferred by analysis.

Definitie 2-5, Logical Coupling (D'Ambros, Lanza, & Lungu, 2006)

Als we dit proberen te illustreren, resulteert dit in het volgende plaatje (***Figuur 2-4***).



Figuur 2-4, Voorstelling Logical Couplings

Zoals in ***Figuur 2-4*** wordt geïllustreerd; ligt er tussen elke combinatie van componenten een Logical Coupling wanneer we uitgaan van de voorgaande definities. Bovengenoemde definities hebben echter een aantal beperkingen. Niet elke Change Dependency staat gelijk aan een Logical Coupling. Omdat Logical Couplings inhoudelijk wijzigingen zijn en een wijziging ook van toepassing kan zijn op niet-

inhoudelijke elementen (***Voorbeeld 2-2***). In dat geval is er wel een Change Dependencie maar geen Logical Coupling.

Voorbeeld 2-2:

Wanneer een taak die aan een applicatie doorgevoerd moet worden de volgende is:

Alle commentaar binnen de applicatie dient gereviewd te worden.

Zullen alle services waarin het commentaar aangepast wordt een Change Dependencie met elkaar vormen. Echter op inhoudelijk vlak hebben er geen wijzigingen plaats gevonden. Er kan dan ook niet gesproken worden van een Logical Coupling.

Het formuleren van een juiste definitie voor het begrip Logical Coupling is lastig omdat er verschillende voorbeelden zijn van gevallen waarin een impliciete afhankelijkheid optreedt. De omschrijving die nog het meest treffend is, is de volgende:

- Een software engineer weet dat wanneer hij een bepaald type wijziging wil doorvoeren hij bepaalde componenten dient te wijzigen (Gall, Jazayeri, & Krajewski, 2003).

De definitie voor Logical Couplings die tijdens het onderzoek gehanteerd zal worden is de volgende:

Logical couplings zijn inhoudelijk afhankelijkheden tussen verschillende componenten die door een expert geconstateerd kunnen worden en die gezamelijke wijzigingen, tussen de verschillende componenten, kunnen veroorzaken.

Definitie 2-6, Definitie Logical Coupling tijdens onderzoek

Een voorbeeld van een Logical Coupling wordt hieronder (***Voorbeeld 2-3***) beschreven.

Voorbeeld 2-3:

In het geval van **Voorbeeld 2-1** *is er sprake van een inhoudelijke wijziging in de Frond end en de Business service die dezelfde business rule afdwingen en implementeren. Hier is dus sprake van een Logical Coupling. Een dergelijke Logical Coupling is een koppeling die geen problemen oplevert en misschien zelfs onvermijdelijk is. Echter wanneer ook de Business service die de persoonsgegevens beheert gewijzigd dient te worden naar aanleiding van de gewijzigde business rule zit er iets verkeerd in de architectuur en verdient de koppeling verder onderzoek.*

De Change Dependencies kunnen als indicator gebruikt worden om Logical Couplings aan te tonen wanneer we de volgende aanname doen.

Logical Couplings veroorzaken gezamenlijk wijzigingen, tussen de verschillende deelnemers van de Logical Coupling, wanneer er een wijziging plaats vindt in de inhoudelijke afhankelijkheid.

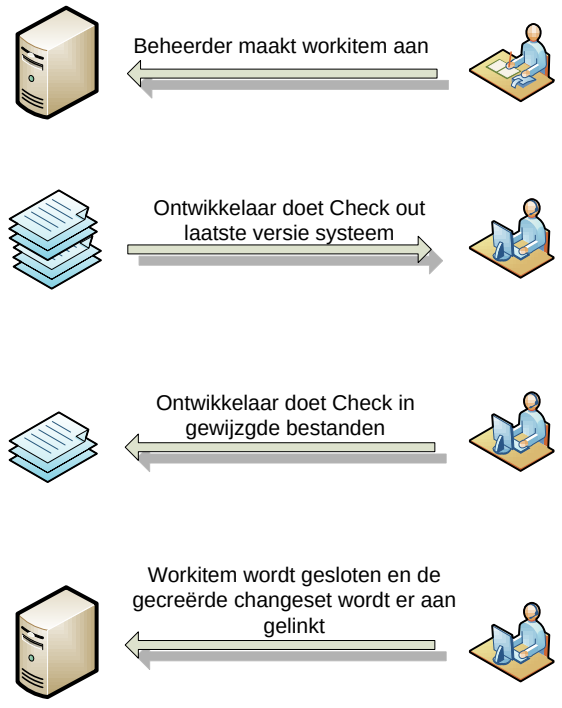
Aanname 2-1, Change Dependencies & Logical Couplings

Een kanttekening bij deze aanname is dat het omgekeerde niet altijd hoeft te gelden. Niet alle gezamenlijke wijzigingen duiden op Logical Couplings.

2.4 | Workitems & Changesets binnen Endeavour

Visual Studio Team Systeem[3] is een belangrijk onderdeel binnen de ontwikkelstraat Endeavour.Net. Visual Studio vormt de ontwikkelomgeving voor de ontwikkelaars wanneer deze werkzaam zijn aan verschillende projecten die binnen de ontwikkelstraat worden gerealiseerd. Daarnaast vindt op de Team Foundation Server de administratie plaats van de verschillende werkzaamheden die aan het systeem dienen uitgevoerd te worden. Ook worden hier de verschillende versies van het systeem in ontwikkeling opgeslagen.

De verschillende werkzaamheden die aan het systeem in ontwikkeling moeten worden uitgevoerd kunnen in de vorm van Workitems geadministreerd worden op de Foundation Server. Een Workitem is een taak van een bepaald type. Deze types kunnen beheerd worden op de server en zelf aangemaakt of gewijzigd worden. Voorbeelden van types zijn; defects, tasks.



Figuur 2-5, Voorstelling workflow Workitem

[3] Voor meer informatie over Visual Studio Team System: http://msdn2.microsoft.com/en-us/teamsystem/default.aspx

Een Workitem wordt door een persoon met de juiste rechten aangemaakt (doorgaans een architect) en toegewezen aan een ontwikkelaar (***Figuur 2-5***). Vervolgens zal de betreffende ontwikkelaar de laatste versie van het systeem binnen halen naar zijn lokale ontwikkelomgeving. Na het Workitem uitgevoerd te hebben worden de gewijzigde bestanden weer terug gezet op de server. Dit resulteert in een Changeset van de gewijzigde bestanden die onder een nieuw versienummer terug te vinden zijn op de Foundation Server. Vervolgens zal de ontwikkelaar het Workitem sluiten en de Changeset linken aan het gesloten Workitem. Vaak is een Workitem niet na 1 wijziging opgelost en hierdoor komt het vaak voor dat aan een Workitem meer dan 1 Changeset is gelinkt.

Binnen Endeavour is ervoor gekozen om elke Changeset maar aan een Workitem te linken zodat van elk Workitem precies na te gaan is welke wijzigingen hebben plaats gevonden. Dit resultaat wordt bereikt door een ontwikkelaar tijdens zijn werkzaamheden aan 1 Workitem tegelijk te laten werken. Hierdoor is elke Changeset maar van toepassing op 1 Workitem. Echter in dit onderzoek zal de ontwikkelde tool aantonen dat deze regels niet altijd nageleeft worden.

2.5 | Vektis Tarieven Orgaan Donatie

Vektis [4] is een bedrijf dat informatie verstrekt aan zorgverzekeraars omtrent de kosten van de gezondheidszorg binnen Nederland. Daarnaast verzorgt Vektis standaardisatie van gegevens uitwisseling tussen zorgaanbieders en zorgverzekeraars. Vaak gaat het hier om tarief informatie.

Info Support is voor Vektis werkzaam aan een project, dat heeft te maken met de tarieven van orgaandonatie. Dit project zal tijdens het onderzoek gebruikt worden om analyse op uit te voeren en de voorgestelde methode te toetsen.

Het project is ten tijde van het onderzoek 4 maanden in ontwikkeling. De grootte en complexiteit van het systeem wordt uitgedrukt in functiepunten[5] aan de hand van een Functie Punt Analyse. Release 1 welke gebruikt wordt in het onderzoek heeft een omvang van 241 functiepunten. Het doel van het project is om de huidige informatievoorziening te moderniseren zodat de input data beter verwerkbaar is en inzichtelijk gemaakt kan worden.

[4] Voor meer informatie over Vektis:
http://www.vektis.nl/index.php?option=com_content&task=view&id=3&Itemid=26
[5] De functiepunten zijn conform ISO/IEC 24570, voor meer informatie over Funtie Punt Analyse(FPA):
http://www.nesma.nl/sectie/home/

3. | Probleemstelling & Motivatie & Inperking

In dit hoofdstuk wordt de probleemstelling en motivatie gegeven voor het onderzochte probleem. Vervolgens worden de grenzen waarbinnen het onderzoek heeft plaats gevonden beschreven.

3.1 | Probleemstelling

Zoals hiervoor uiteengezet is, wordt in de ontwikkelstraat Endeavour een SOA architectuur voorgeschreven. De voorgestelde implementatie dient te voldoen aan bepaalde regels. Deze regels en de voorgestelde implementatie worden beschreven in de Logische Referentie Architectuur (LRA)(de Vries, ENDEV01 - Logical Reference Architecture, 2006). De gehele ontwikkelstraat heeft als doel de productiviteit en effectiviteit van software ontwikkeling te verbeteren ten opzichte van andere ontwikkel methoden. Binnen de ontwikkelstraat is er gekozen voor een SOA omdat deze flexibel en onderhoudsvriendelijk zou moeten zijn en een minimum aan koppeling als voordeel heeft.

Doordat een SOA bestaat uit autonoom functionerende services is het moeilijk om te monitoren of er bepaalde afhankelijkheden ontstaan in de vorm van Logical Couplings tussen verschillende Ci's. Logical Couplings kunnen duiden op problemen wanneer de gestelde architectuurprincipes worden overtreden. Dit wordt zichtbaar doordat er steeds verschillende CI's in combinatie met elkaar gewijzigd moeten worden die normaliter volledig los van elkaar horen te staan. Dergelijke problemen maken het onderhoud van een systeem duur en gecompliceerd. Wanneer verschillende afhankelijkheden, in de vorm van Logical Couplings, in de architectuur aangetoond kunnen worden, kan er onderzoek gedaan worden naar een mogelijkheid om deze problemen te verhelpen of zelfs te voorkomen. Hieruit ontstaat de volgende (***Tabel 3-1***) hoofdvraag.

Hoe kunnen Logical Couplings binnen een SOA architectuur, van systemen ontwikkeld binnen Endeavour, aangetoond en inzichtelijk gemaakt worden.

Tabel 3-1, Hoofdvraag

Om deze hoofdvraag te kunnen beantwoorden dienen verschillende deelvragen beantwoord te worden. De gestelde deelvragen worden hieronder (***Tabel 3-2***) beschreven.

Indicator	Deelvraag
DV1	Welke methode kan gebruikt worden om Logical Couplings aan te tonen?
DV2	Hoe kan deze methode toegepast worden binnen de omgeving van Endeavour?
DV3	Hoe kunnen de Logical Couplings inzichtelijk gemaakt worden?

Tabel 3-2, Deelvragen

3.2 | Alternatieven: Statische & Dynamische analyse

Voor het aantonen van afhankelijkheden en koppelingen wordt vaak statische of dynamische analyse gebruikt. Echter hebben deze methoden verschillende nadelen waardoor ze niet toereikend zijn om in de huidige situatie te functioneren. De verschillende nadelen worden hieronder toegelicht.

Calls zijn geminimaliseerd

Zowel statische als dynamische analyse toont directe afhankelijkheden aan in de vorm van calls tussen verschillende componenten. Omdat het onderzoek zich richt op een SOA architectuur zullen dergelijke analyses weinig nuttige informatie opleveren omdat een SOA architectuur dergelijke afhankelijkheden namelijk minimaliseert. Daarnaast willen we afhankelijkheden aantonen tussen verschillende services die in een SOA omgeving volledig onbekend voor elkaar zijn.

Inhoudelijke afhankelijkheden

Zowel statische als dynamische analyse toont directe afhankelijkheden aan in de vorm van calls tussen verschillende componenten, terwijl het onderzoek zich richt op afhankelijkheden die niet in de vorm van calls zichtbaar zijn. Het gaat hier om inhoudelijke afhankelijkheden, zogenaamde Logical Couplings.

Tijdrovend proces

Wanneer statische analyse wordt toegepast dient de gehele source code geanalyseerd en vervolgens geïnterpreteerd te worden om de afhankelijkheden te bepalen. Bij dynamische analyse dienen de verschillende functionaliteiten van het programma uitgevoerd te worden om te kunnen analyseren waar de afhankelijkheden zitten. Een analyse met beide methoden is een tijdrovende klus.

3.3 | Scope

Binnen het onderzoek is een bepaalde scope beschreven waardoor het onderzoek op bepaalde onderdelen afgebakend is. Dit is gedaan om te voorkomen dat het onderzoek te omvangrijk zou zijn om in de korte tijd te doorlopen. Hieronder worden de grenzen aangegeven waarbinnen dit onderzoek is uitgevoerd.

Endeavour .Net

Zoals al eerder genoemd is de Ontwikelstraat in twee smaken beschikbaar; .Net en Java. De .Net-variant maakt gebruik van Visual Studio 2005 als ontwikkelomgeving; de Java-variant daarentegen gebruikt Eclipse. Omdat de Java-variant van de ontwikkelstraat nog maar kort in gebruik is, was er geen project beschikbaar dat gebruikt kon worden voor het uiteindelijke onderzoek. Daarom is de .Net-variant gekozen.

Source base: Team Server 2005

Hier is gekozen voor Team Server 2005 omdat deze de mogelijkheid geeft om bepaalde informatie te gebruiken in de analyse, wat in het geval van Source Safe niet mogelijk is. Deze informatie bestaat uit Workitems en Changesets (***2.4| Workitems & Changesets binnen Endeavour***) welke gebruikt kunnen worden om bepaalde wijzigingen te relateren aan gewijzigde bestanden. In het geval van Source Safe is dit niet aan elkaar gelinkt en ontbreekt deze informatie. Daarnaast is het voor de korte tijd niet mogelijk om het Proof of Concept (PoC) support te laten leveren voor meer dan één source base.

Administratieve systemen

Tijdens het onderzoek ligt de focus op administratiefe systemen omdat Endeavour hiervoor ontwikkeld is. De Logische Referentie Architectuur (***2.1.5| Logische Referentie Architectuur***) is dan ook van toepassing op dit soort systemen. Tijdens het onderzoek heeft het "Vektis Tarieven Orgaan Donatie" systeem gediend als het systeem waar de voorgestelde methode in hoofdstuk ***4| LCA – Een Tool voor het analyseren van Logical Couplings*** op is toegepast.

4. | LCA – Een Tool voor het analyseren van Logical Couplings

In dit hoofdstuk wordt de methode die gebruikt is tijdens het onderzoek voor het oplossen van het probleem, beschreven in hoofdstuk ***3| Probleemstelling & Motivatie & Inperking***, uiteengezet worden. Er worden verschillende hypotheses geformuleerd die in het uiteindelijke onderzoek getest zijn. Vervolgens wordt de aanpak van het onderzoek beschreven.

4.1 | Methode

Om de hoofdvraag gesteld in hoofdstuk ***3| Probleemstelling & Motivatie & Inperking*** te beantwoorden dienden eerst de verschillende deelvragen beantwoord te worden. Deelvraag ***DV1*** heeft betrekking op de methode die gekozen is om de Logical Couplings aan te tonen. De methode die toegepast is in het onderzoek is gebaseerd op een methode die voorgesteld is door (Gall, Jazayeri, & Krajewski, 2003).

4.1.1 | Bestaande Methode

De methode maakt gebruik van de Release History van een systeem die meestal opgeslagen is in een source base. De methode is retrospectief, wat wil zeggen dat zij zich richt op verschillende meetpunten in de tijd. Door de wijzigingen van verschillende tijdpunten uit een source base te halen kunnen analyses worden gedaan. (Gall, Jazayeri, & Krajewski, 2003)

De analyse richt zich op de werkelijke source code die in de source base is terug te vinden en laat de verschillende resource bestanden zoals configuratie, images etc. buiten beschouwing. Vervolgens wordt er gebruik gemaakt van het feit dat er Strong Ownership van toepassing is op de code om de change sets van de verschillende wijzigingen samen te stellen. Vervolgens maakt men onderscheid tussen Internal couplings (links) en External couplings (links). Internal couplings zijn koppelingen die binnen een bepaalde module of zijn submodules plaats vinden. External couplings zijn koppelingen die tussen een bepaalde module (of zijn submodules) en een andere module (of zijn submodules) van het systeem plaats vinden.(Gall, Jazayeri, & Krajewski, 2003)

Wanneer de verschillende afhankelijkheden in kaart zijn gebracht kunnen er verschillende vragen worden beantwoord (Gall, Jazayeri, & Krajewski, 2003). Hieronder (***Tabel 4-1***) worden enkele voorbeelden gegeven.

Vragen:
Welke delen van het systeem zijn het meest tegelijk gewijzigd
Hoeveel classes zijn inbegrepen in een external coupling naar een andere module
Is de internal of de external coupling sterker

Tabel 4-1, Vraagstukken naar aanleiding van resultaten van methode

4.1.2 | Voor- & Nadelen Methode

De door Gall, Jazayeri, & Krajewski voorgestelde aanpak heeft verschillende voordelen welke hieronder (*Tabel 4-2*) toegelicht worden.

Voordelen:
Er hoeft geen analyse te worden uitgevoerd op source code
Logical dependencies kunnen worden aangetoond

Tabel 4-2, Voordelen gebruik van Release History

Naast de voordelen zijn er ook verschillende nadelen welke hieronder (*Tabel 4-3*) toegelicht worden.

Nadelen:
Er dienen eerst wijzigingen gemaakt te worden alvorens Change Dependencies aangetoond kunnen worden
Er dient Strong Ownership van de code te zijn, een taak dient door één programmeur uitgevoerd te worden om de change set te kunnen bepalen **(Gall, Jazayeri, & Krajewski, 2003)**
De code dient na elke unieke taak ingecheckt te worden om een gedegen release history op te bouwen die gebruikt kan worden voor het onderzoek

Tabel 4-3, Nadelen gebruik van Release History

4.1.3 | Toepasbaarheid methode

Binnen de ontwikkelomgeving van Endeavour wordt gewerkt met Workitems. In deze Workitems worden de taken die uitgevoerd dienen te worden op het systeem gespecificeerd. Voor de toepasbaarheid van de methode op systemen ontwikkeld binnen Endeavour is de volgende aanname gedaan:

In de toe te passen methode wordt een unieke wijziging gezien als een Workitem dus een taak doorgevoerd aan het systeem.

Aanname 4-1, Unieke wijziging

De laatste twee nadelen, genoemd in de vorige paragraaf, zouden onder deze aanname dus niet van toepassing moeten zijn. Doordat de Changesets gekoppeld zijn aan een Workitem kunnen verschillende mensen zolang als ze willen aan de taak werken. Elke Changeset wordt vervolgens gekoppeld aan het desbetreffende Workitem. De gewijzigde bestanden zijn vervolgens afleidbaar. In plaats van Strong Ownership om de Changeset te bepalen kunnen de Changesets nu gewoon opgevraagd worden. Inchecken hoeft niet per taak omdat de Changeset niet meer bepaald hoeft te worden, maar er meer dan 1 Changeset gekoppeld kunnen worden aan een Workitem.

4.2 | Hypothesen

Naar aanleiding van het literatuuronderzoek is er een bepaalde hypothese opgesteld welke als uitgangspunt diende voor het onderzoek. De gestelde hypothese wordt hieronder (*Hypothese 4-1*) beschreven.

Binnen Endeavour geven Change Dependencies een goede indicatie van de Logical Couplings die aanwezig zijn in de systemen in ontwikkeling.

Hypothese 4-1, Change Dependencies als indicator voor Logical Couplings

Later in het onderzoek (***6| Resultaten***) blijkt dat binnen Vektis deze hypothese niet opgaat door de verkeerde inrichting van de source base en de grote aantallen aan False-positives.

4.3 | Aanpak

De gekozen aanpak voor het onderzoek bestaat uit twee onderdelen die overeenkomen met de gestelde deelvragen ***DV2*** en ***DV3***; de ontwikkeling van een PoC, validatie van PoC en toegepaste methode. Beide onderdelen worden hieronder toegelicht.

4.3.1 | Ontwikkeling Proof of Concept

Omdat er geen bestaande tooling beschikbaar was die in de te onderzoeken omgeving toegepast kon worden om de voorgestelde methode te onderzoeken, is er een PoC ontwikkeld. Op de eerste plaats is het doel van het PoC om te onderzoeken hoe we de voorgestelde methode kunnen toepassen binnen Endeavour (***DV2***). Daarnaast kon aan de hand van het PoC de gestelde hypotheses getoetst worden en het onderzoek daadwerkelijk uitgevoerd worden. Vervolgens kon aan de hand van het PoC deelvraag 3 (***DV3***) beantwoord worden. Aan de hand van het PoC kon de voorgestelde methode onderzocht worden. Daarnaast kon met behulp van het PoC bepaalde worden of de methode daadwerkelijk een toegevoegde waarde zou leveren binnen de bestaande kwaliteitscontrole van Endeavour.

4.3.2 | Validatie

Omdat er hier een tool ontwikkeld is, bestaat de eerste stap van de validatie uit het toetsen van de juistheid van de resultaten die de tool produceert. De volgende stap zal zijn om de toegevoegde waarde van de tool te onderzoeken en te valideren.

4.3.2.1 | Toetsing Proof of Concept

Om te valideren dat het PoC de juiste resultaten toont, is deze getoetst. Dit is gedaan aan de hand van een aantal criteria. Deze tezamen geven aan hoe accuraat de tool functioneert. De criteria worden hieronder (***Tabel 4-4***) toegelicht.(Relevance, Recall, Precision, Fallout, 2000)

Criteria:	Omschrijving:
Recall	Hoe goed kunnen de Logical Couplings die aanwezig zijn met de toegepaste methode daadwerkelijk ook aangetoond worden. *Vb, wanneer na een run van de tool maar 80 koppelingen van de 1000 aantoonbare*

	*koppelingen aangetoond worden is de recall 8% (80/1000*100= 8%)*
Precision	Hoe precies worden de echt Logical Couplings die aanwezig zijn en aantoonbaar zijn aangetoond? Worden er ook niet relevante koppelingen aangetoond? Of worden er bepaalde relevante koppelingen niet aangetoond? Met andere woorden worden er false-positives of false-negatives gegenereerd? *Vb, wanneer na een run van de tool 1000 koppelingen worden aangetoond waarvan er uiteindelijk maar 20 relevant blijken omdat ze ook daadwerkelijk Logical Couplings zijn dan is de precision 2% (20/1000*100=2%)* Het tegenovergestelde van Precision is de **Fallout;** dit is dus het percentage dat uit false-positives bestaat.

Tabel 4-4, Validatie criteria PoC (Relevance, Recall, Precision, Fallout, 2000)

De toetsing van het PoC kan in twee stappen verdeeld worden. De eerste stap wordt gerealiseerd door de tool op een kleine voorbeeld source base, waarvan de afleidbare Logical Couplings bekend zijn, een analyse uit te laten voeren. Vervolgens kunnen de verschillende criteria op de resultaten onderzocht worden. In stap twee kunnen bestaande systemen onderzocht worden en de resultaten getoetst door steekproeven te doen welke aan de hand van interviews doorgesproken worden met de verantwoordelijke experts.

4.3.2.2 |Validatie resultaten Methode

Om te valideren dat de resultaten die de methode levert een meerwaarde leveren op de bestaande kwaliteitscontrole zijn de meningen van experts verkregen door interviews gebruikt. Deze stap en het bespreken van de steekproeven, voor de toetsing van het PoC, zijn dan ook redelijk simultaan verlopen.

De mogelijkheid bestond dat uit de meningen van de experts zou blijken dat de resultaten in zijn huidige vorm geen toegevoegde waarde hebben. Wanneer dit het geval zou zijn zouden de volgende vragen gesteld kunnen worden om te onderzoeken wat de oorzaak was.

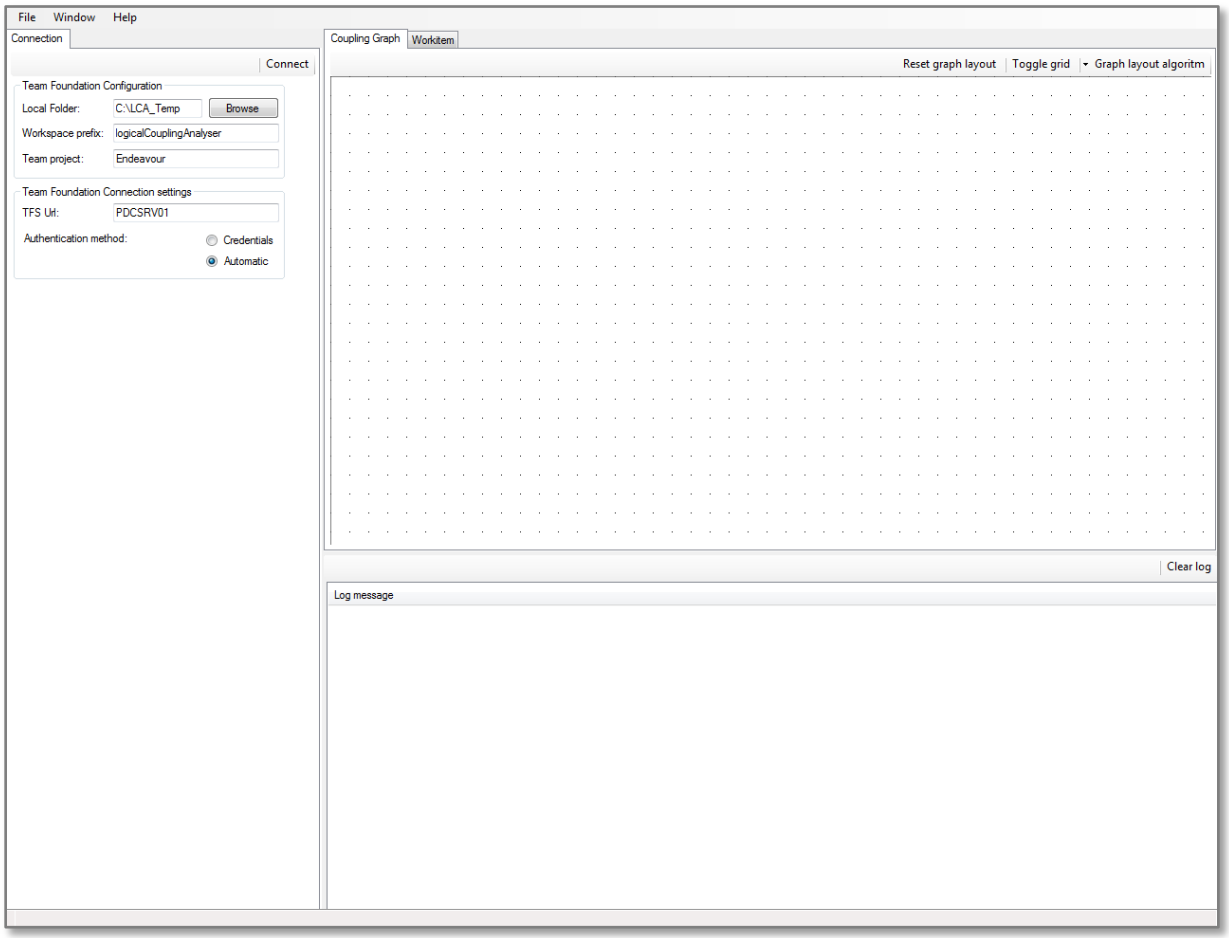
- Zijn de resultaten onbruikbaar omdat er geen conclusies aan verbonden kunnen worden en omdat de resultaten geen mogelijkheid geven tot verdere analyse van het systeem?
- Zijn de resultaten onbruikbaar omdat de input, in de onderzochte situatie, vervuild is en daardoor ongeschikt om bruikbare resultaten te genereren?

5. | Toepassing van LCA op Vektis

In dit hoofdstuk wordt het onderzoek beschreven dat is uitgevoerd. Daarnaast worden de verschillende hypotheses die als uitgangspunt voor het onderzoek hebben gediend, besproken. Als laatste wordt de validatie van het PoC en de methode besproken.

5.1 | Ontwikkeling LCA

De Logical Coupling Analyser[6] (LCA)(***Figuur 5-1***) is de uiteindelijke tool die gerealiseerd is tijdens het onderzoek. De LCA is een windows client applicatie die het mogelijk maakt om de Change Dependencies die aanwezig zijn in een Team project opgeslagen op een Team Foundation Server, of zijn opvolger ORCAS, in kaart te brengen en inzichtelijk te maken. Aan de hand van dit PoC is de voorgestelde methode uitgevoerd en getest op een bestaand systeem (Vektis).



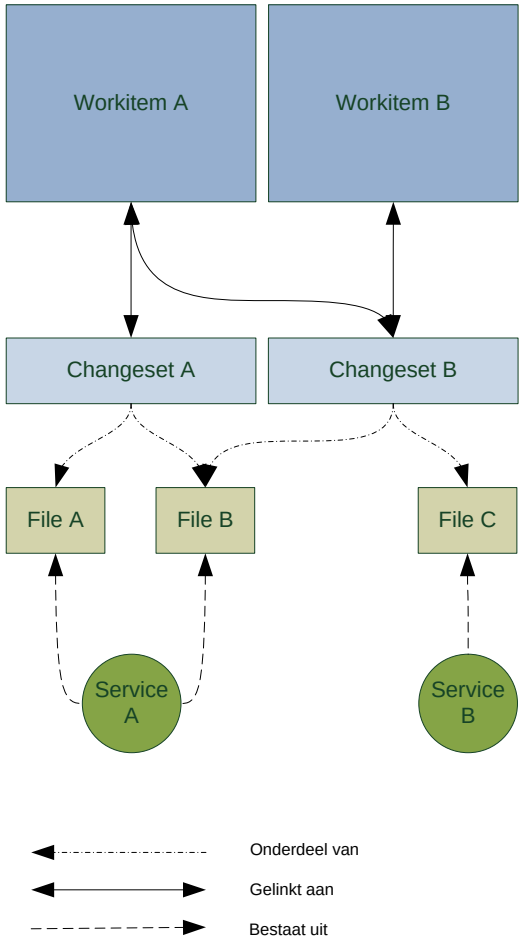
Figuur 5-1, Logical Coupling Analyser (LCA)

[6] Voor meer informatie over de LCA: ***Bijlage A***

Alvorens de LCA ontwikkeld kon worden is gekeken naar hoe de voorgestelde methode toegepast kon worden op systemen ontwikkeld binnen Endeavour.

5.1.1 | Aannames

Binnen Endeavour .Net wordt gewerkt met de IDE (Integrated Development Environment) Visual Studio 2005. Daarnaast wordt ook gebruik gemaakt van een buildserver 'Visual Studio Team Server' waar alle bestanden worden opgeslagen. De wijzigingen gemaakt op elke versie zijn hier opgeslagen en maken deel uit van een bepaalde Changeset. De taken die op het systeem worden uitgevoerd zijn in de vorm van Workitems opgeslagen op de team server. In theorie kan een Workitem verschillende Changesets bevatten en een Changeset kan weer aan meer dan 1 Workitem gelinkt zijn. In ***Figuur 5-2*** wordt een voorstelling gegeven van hoe een mogelijke situatie eruit zou kunnen zien.



Figuur 5-2, Voorstelling van de mogelijke structuur van een Workitem, Changeset koppeling.

Echter deze mogelijkheid maakt de analyse moeilijk omdat hierdoor niet nagegaan kan worden op welke services een Workitem van toepassing is. De voorgestelde situatie bevat de volgende Change Dependencies:

- Workitem A; *Service A <-> Service B*
- Workitem B; *Service A <-> Service B*

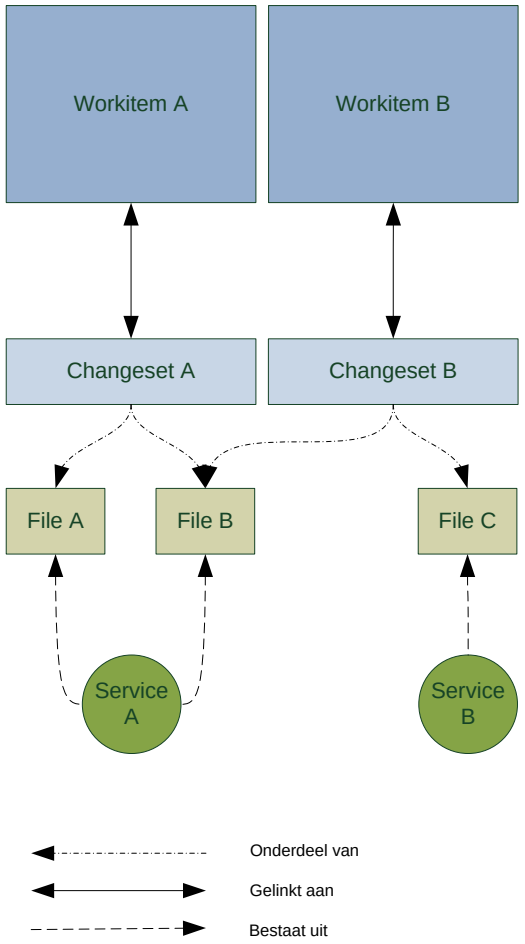
Echter is niet duidelijk of **File C** gewijzigd is voor **Workitem A** of **Workitem B**. Daardoor wordt de zekerheid van de Change Dependencies tussen de verschillende services ook meteen onzeker. In het geval dat **File C** gewijzgd is voor **Workitem B** vallen alle Change Dependencies tussen de services weg.

Daarnaast is vanuit Info Support binnen Endeavour aangegeven dat men altijd aan 1 Workitem tegelijk werkt (***2.1.3| Endeavour***) waardoor de volgende aanname gedaan kan worden:

Elk Workitem bevat 1 of meer Changesets en elke Changeset is maar aan 1 Workitem gelinkt.

Aanname 5-1, Workitems & Changesets

Deze aanname is in ***Figuur 5-3*** afgebeeld.



Figuur 5-3, Voorstelling van de aanname dat elke Changeset maar aan een Workitem is gekoppeld.

De Change Dependencies die nu zichtbaar worden, zijn de volgende:

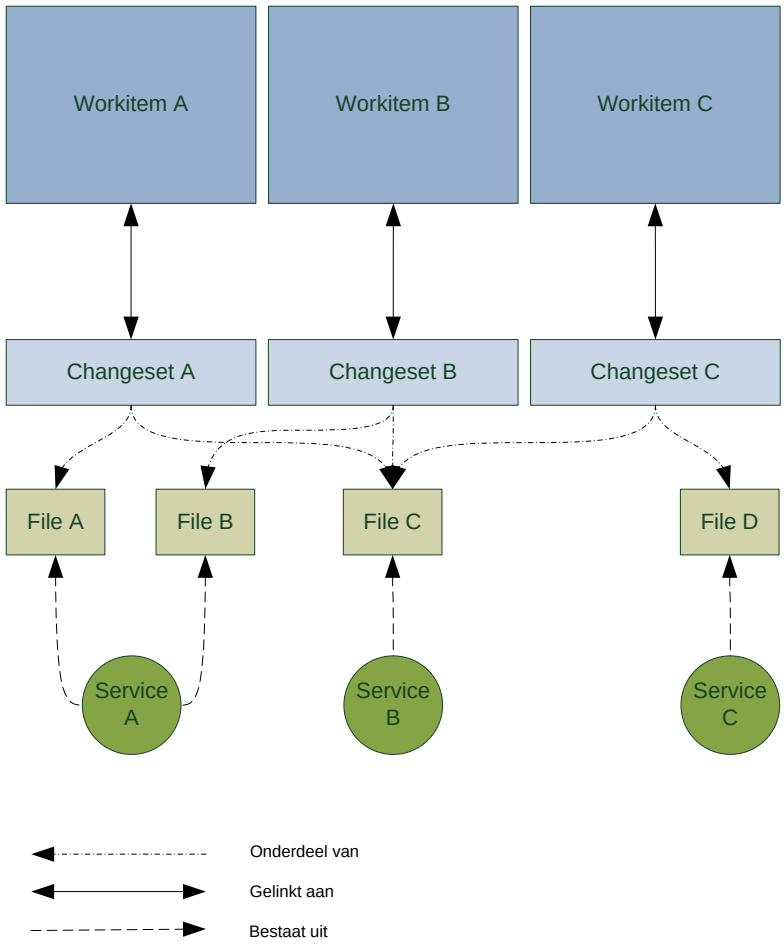
- Workitem A; *Geen*
- Workitem B; *Service A <-> Service B*

5.1.2 | Implementatie methode

De LCA kan gebruikt worden om onderzoek te doen op een Team project opgeslagen op een Team Foundation Server of zijn opvolger Orcas. Dit doet hij door een extractie te maken van de Workitems en vervolgens alle Change Dependencies te bepalen. Dankzij ***Aanname 5-1*** is het mogelijk om Change Dependencies tussen services af te leiden uit de Workitems die uitgevoerd zijn op het systeem in ontwikkeling. In de Changesets bevinden zich de bestanden die aangepast zijn. De bijbehorende service kan afgeleid worden uit het pad van het bestand. Doordat bekend is aan welk

Workitem een Changeset gelinkt is, is indirect ook bekend welke services aan het Workitem gelinkt zijn (***Figuur 5-3***).

Om de graad van afhankelijkheid te bepalen is er voor gekozen om voor elke combinatie van services deze gelijk te stellen aan het aantal Change Dependencies waarin de combinatie voorkomt.



Figuur 5-4, Voorstelling bepaling graad van afhankelijkheid

Voorbeeld;

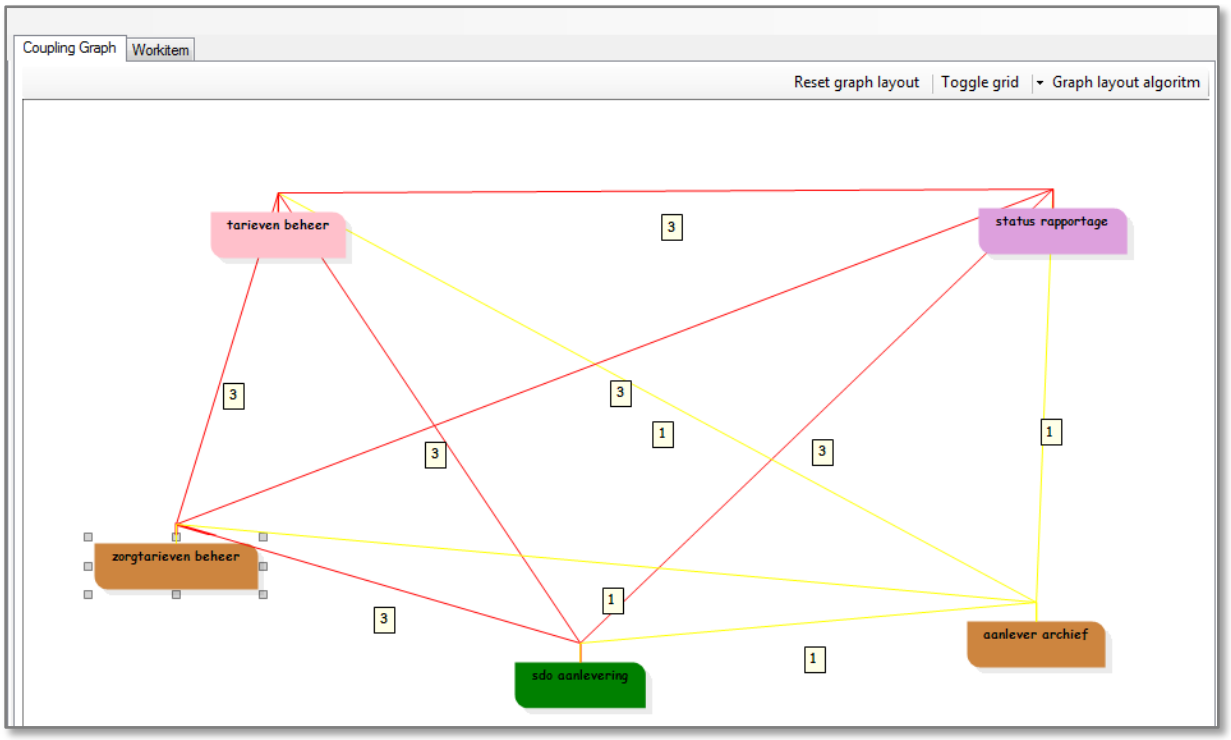
In ***Figuur 5-4*** zijn de volgende Change Dependencies zichtbaar:

- Workitem A; *Service A <-> Service B*
- Workitem B; *Service A <-> Service B*
- Workitem C; *Service B <-> Service C*

Dit resulteert in de volgende graad van afhankelijkheid per combinatie van services:

- Service A <-> Service B; *graad van afhankelijkheid: 2 (Workitem A en B)*
- Service B <-> Service C; *graad van afhankelijkheid: 1 (Workitem C)*

Voor de weergave van de Change Dependencies is voor een graaf gekozen. Een graaf geeft een goed overzicht van de verschillende Change Dependencies die aanwezig zijn door de services af te beelden en deze met elkaar te verbinden.



Figuur 5-5, LCA; graaf toont Change Dependencies extractie Vektis

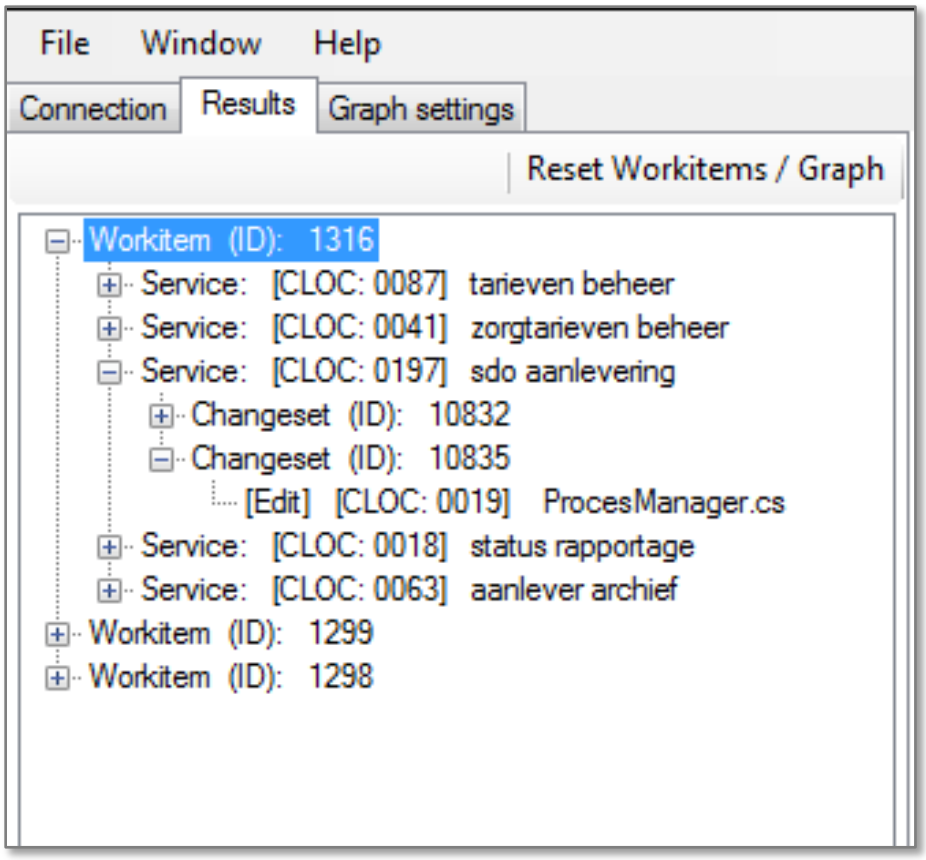
In ***Figuur 5-5*** is zichtbaar hoe de LCA de verschillende CI's in het team project toont. Elk type service heeft een andere kleur om makkelijk onderscheid te kunnen maken. De graad van afhankelijkheid van de verschillende combinaties van services is zichtbaar naast de verbindende lijnen. Daarnaast wordt er gekeken naar wat de minima en maxima zijn om de lijnen een kleur te geven:

- Groen (minima)
- Genivileerd (tussenliggende waarden)
- Rood (maxima)

Het gebruik van de verschillende kleuren voor de verschillende typen services en verbindingslijnen maakt het mogelijk om bepaalde hotspots van ongewenste afhankelijkheden in een oogopslag op te merken.

Om een Change Dependency verder te kunnen onderzoeken is er voor gekozen om de Workitems, waar de Change Dependencies uit bestaan, in een treeview weer te geven. Deze view maakt

het mogelijk om voor elk Workitem de Services te bekijken en de bijbehorende gewijzigde bestanden (gegroepeerd op Changeset).

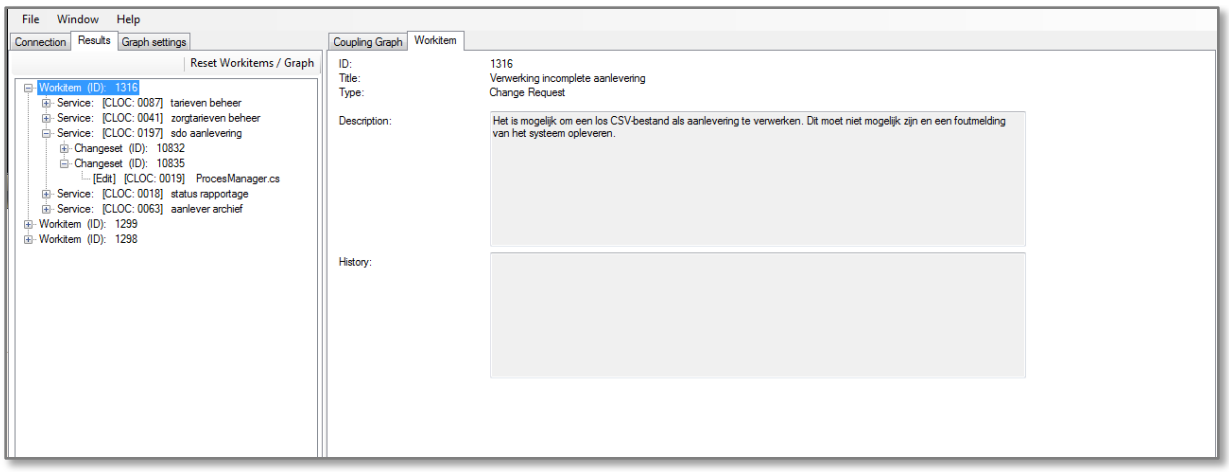


Figuur 5-6, LCA; treeview toont Workitems extractie Vektis

In ***Figuur 5-6*** zijn de verschillende Workitems die zich in de extractie bevinden zichtbaar. Workitem 1316 is van toepassing geweest op de CI's:

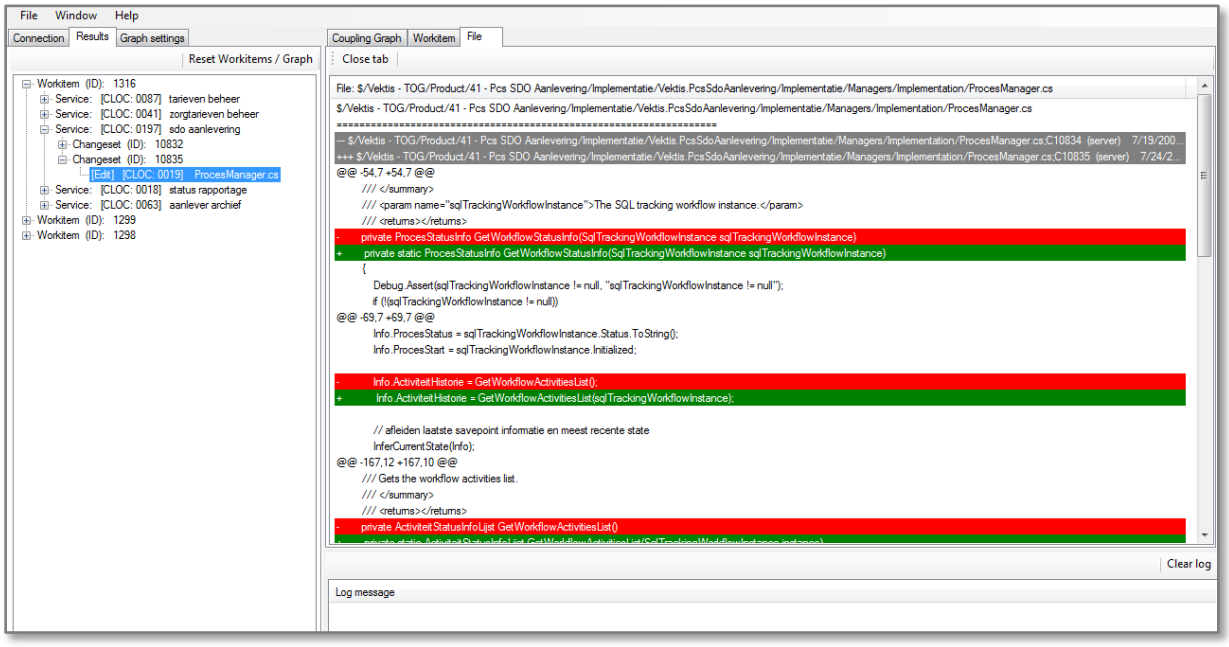
- Tarieven beheer
- Zorgtarieven beheer
- Sdo aanlevering
- Status rapportage
- Aanlever archief

In het CI sdo aanlevering zijn in totaal 197 Changed Lines Of Code (CLOC) waarvan er 19 zijn gewijzigd in het bestand ProcesManager.cs. Ook kan het Workitem bekeken worden zodat de detaills doorgenomen kunnen worden.



Figuur 5-7, LCA; Workitem 1316 details worden getoond

Uiteindelijk is het doel om Logical Couplings aan te tonen. Daarom kan van elk bestand dat gewijzigd is de filediff bekeken worden. Dit geeft de mogelijkheid om elke Change Dependency uitvoerig te analyseren.



Figuur 5-8, LCA; filediff toont wijzigingen ProcesManager.cs

In Figuur 5-8 is de filediff zichtbaar van het bestand ProcesManager.cs.

5.2 | Toetsing Proof of Concept

Zoals eerder beschreven in ***4| LCA – Een Tool voor het analyseren van Logical Couplings*** diende de ontwikkelde LCA in de vorm van een PoC op de eerste plaats getoetst te worden op juistheid van de resultaten. In deze paragraaf zal de toetsing die uitgevoerd is op de LCA beschreven worden.

5.2.1 | Toetsing Scenario's

Om aan te tonen dat de LCA op de juiste manier functioneert zijn er verschillende scenario's (*Tabel 5-1*) opgesteld welke doorlopen zijn met de LCA. De scenario's worden hieronder besproken.

Scenario:
Contact maken met een Team Foundation Server
Workitems ophalen die voldoen aan filter instellingen
Service herkennen
Change Dependencies tonen
File diff tonen
Filter instellingen na extractie toepassen

Tabel 5-1, Toetsing scenario's voor LCA

5.2.1.1 | Contact maken met een Team Foundation Server

Alvorens een extractie kan worden gedaan door de LCA dient deze een connectie te hebben met een Team Foundation Server waar het te onderzoeken Team Project zich bevindt.

5.2.1.2 | Workitems ophalen die voldoen aan filter instellingen

Alvorens een extractie gedaan kan worden kunnen verschillende instellingen gedaan worden. Door voordat een extractie daadwerkelijk wordt uitgevoerd een filter toe te passen kunnen er verschillende extracties worden samengesteld. Deze filter heeft betrekking op drie onderdelen:

- Het type Workitem; task, defect etc.
- Het type bestand dat gewijzigd is
- Het soort wijziging; change, add etc.

Door op deze manier te filteren kunnen False-positives van te voren worden uitgesloten doordat bepaalde misleidende factoren kunnen worden uitgesloten van de extractie. Onder misleidende factoren verstaan we hier:

- Type Workitems die op het gehele systeem van toepassing zijn bijv. 'tasks'
- Bestanden die niet met het systeem te maken hebben bijv. '.doc'
- Wijzigingstypen die een verstoord resultaat kunnen opleveren zoals bijv. 'merge'

Hieronder (*Tabel 5-2*) worden de onderdelen van de filter en hun resultaat toegelicht.

Filter:	Resultaat:
Type Workitem	Door het instellen van deze filter worden alleen Workitems van het geselecteerde type meegenomen in de extractie. Voorbeeld; *Dit geeft de mogelijkheid om bijvoorbeeld alleen het Workitem*

	type 'defects' te analyseren en False-positives naar aanleiding van bijvoorbeeld 'tasks', die doorgaands betrekking hebben op het gehele systeem, weg te filteren.
Type bestanden	Door deze filter te gebruiken worden de Logical Couplings alleen afgeleid uit Workitems met Changesets waarin bestanden van dit type zijn gewijzigd. Voorbeeld; *Hier kan gekozen worden om bijvoorbeeld alleen xml en cSharp bestanden te onderzoeken*
Type wijziging	Door deze filter toe te passen wordt er gekeken of een bestand dat gewijzigd is of het type van die wijziging dan overeenkomt met het type dat ingesteld is. Voorbeeld; *Hier kan gekozen worden om alleen bestanden die wijzigingstype 'Edit' of 'Add' hebben te onderzoeken*

Tabel 5-2, Filter opties voor extractie

5.2.1.3 | Service herkennen

De service die aangepast is wordt afgeleid uit het server pad van een gewijzigd bestand. Daarnaast wordt ook, uit het zelfde pad, afgeleid om wat voor type service het hier gaat. Hieronder (***Tabel 5-3***) is een voorbeeld gegeven van de directory structuur zoals deze voorgeschreven wordt in de Technische Referentie Architectuur (TRA) (de Vries, ENDEV02 - Technical Reference Architecture, 2006).

[pad op systeem] \ [volgnummer] – [CI code] [Service naam] \ Implementation \ [Solution van CI]

Tabel 5-3, Structuur pad van een service (CI)

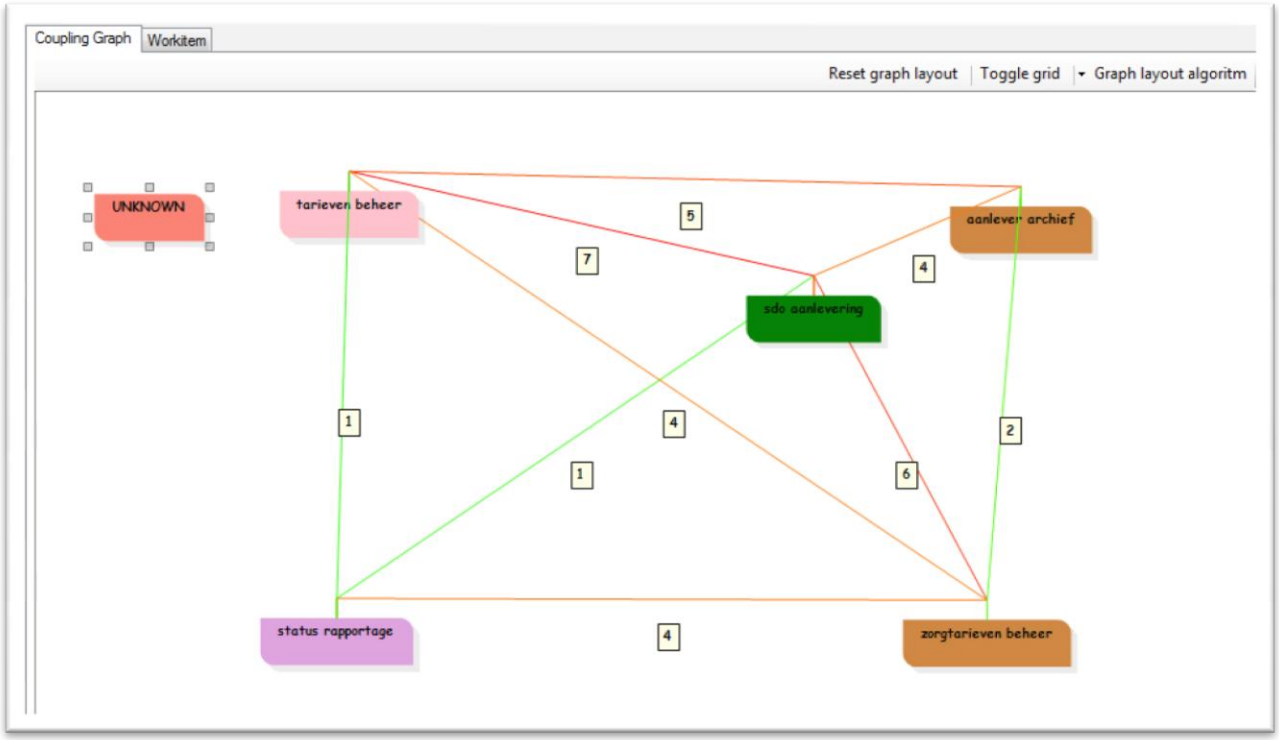
De Configuration Items (CI's) worden op de volgende (***Tabel 5-4***) manier afgekort in de directory structuur verwerkt.

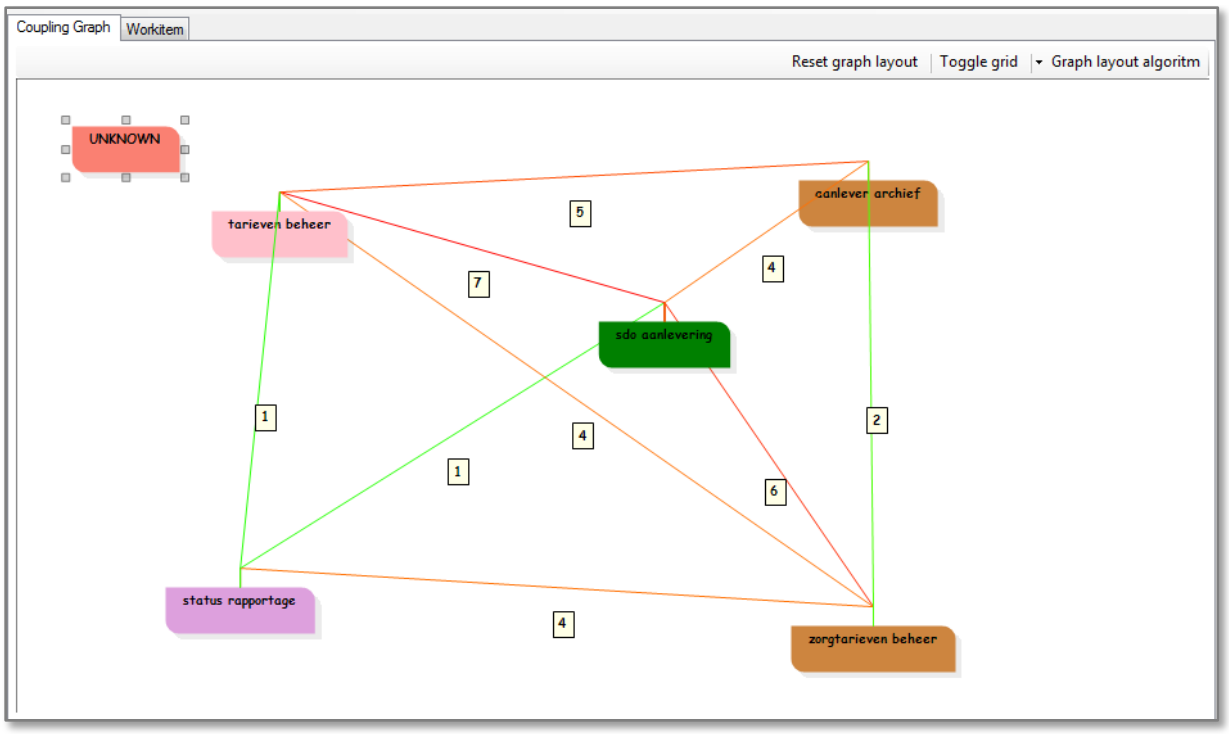
Configuration Item :	Code:
Frond end	FE
Business Services	BS
Integration Services	IS
Proces Services	Pcs
Platform Services	Pfs
Service Bus	SB
Infrastructure Components	IC

Tabel 5-4, Configuration Item code's

5.2.1.4 | Change Dependencies tonen

De Change Dependencies worden aangetoond aan de hand van de bestanden die gewijzigd zijn in de Changesets die gelinkt zijn aan de verschillende Workitems. Wanneer verschillende services zijn gewijzigd onder een en hetzelfde Workitem is er een Change Dependencies aanwezig. De verschillende Change Dependencies worden getoond in een graaf (***Figuur 5-9***). Het gewicht van de koppeling hangt af van hoe vaak de combinatie van services voorkomt bij verschillende Workitems.





Figuur 5-9, LCA toont Change Dependency graaf

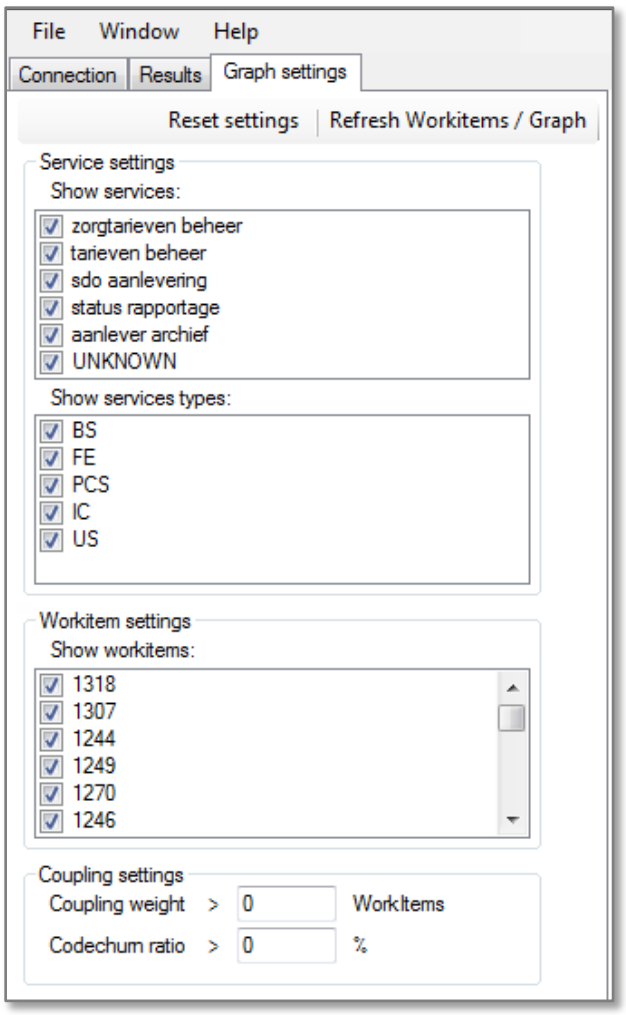
5.2.1.5 | File diff tonen

Om een Change Dependencies verder te onderzoeken kan men de Workitems van de Change Dependency bekijken. Van elk Workitem worden de Changesets en bijbehorende bestanden getoond. Door op een bestand te klikken wordt de file diff getoond. Hierdoor wordt de Code Churn zichtbaar op het bestand (*Figuur 5-11*).

5.2.1.6 | Filter instellingen na extractie toepassen

Door na de extractie te filteren wordt men in staat gesteld de resultaten te verfijnen. Deze filter (*Figuur 5-10*) bevat de volgende mogelijkheden:

- Service tonen/verbergen
- Type service tonen/verbergen
- Workitem tonen/verbergen
- Koppelingen van af een bepaald gewicht tonen
- Koppelingen van af een bepaald Code Churn ratio tonen



Figuur 5-10, LCA filter, na extractie, mogelijkheden

Door deze filter kan men het aantal False-positives weg filteren die zich nog in de extractie bevinden. Hieronder (*Tabel 5-5*) worden de verschillende onderdelen van de filter en hun resultaat toegelicht.

Filter:	Resultaat:
Service tonen/verbergen	Deze filter maakt het mogelijk om bepaalde services weg te filteren. Bijbehorende Logical Couplings worden dan automatisch ook weg gefilterd. Deze filter kan worden gebruikt naar aanleiding van een verkeerd geëxtraheerde service of een overkoepelende service die uiteraard Logical Couplings heeft.
Type service tonen/verbergen	Omdat er in de LRA verschillende regels zijn beschreven hoe verschillende services met elkaar mogen communiceren kan het interessant zijn om bepaalde type services weg te filteren. Hierdoor wordt het resultaat duidelijker en wordt de focus meer gelegd op mogelijke problemen.
Workitem tonen/verbergen	Door deze filter toe te passen kunnen Workitems verborgen worden. Dit kan bijvoorbeeld toegepast worden wanneer een Workitem te algemeen is en

	daardoor te veel betrekking heeft op het gehele systeem.
Coupling weight	Door deze filter te gebruiken worden alleen de koppelingen en Workitems getoond die een groter gewicht hebben dan de ingestelde waarde. Hierdoor is het mogelijk om meer de focus op mogelijke knelpunten te leggen.
Code Churn ratio	Deze filter bekijkt hoe groot het aandeel is van een service in een bepaalde koppeling. Dit wordt gebaseerd op de omvang van de totale wijzigingen in de koppeling en de omvang van de wijzigingen in de specifieke service. Wanneer het aandeel kleiner is dan de ingestelde waarde wordt de koppeling en zo mogelijk het bijbehorende Workitem niet getoond.

Tabel 5-5, Filter opties na extractie

5.2.2 | Resultaten toetsing LCA

Aan de hand van een test source base en test applicatie zijn de verschillende scenario's uitvoerig getest. De voorbeeld applicatie bevatte een aantal services. Door verschillende Workitems aan te maken en vervolgens wijzigingen te introduceren en de verschillende ontstane Changesets te koppelen aan de Workitems konden verschillende toestanden worden nagebootst. Doordat de wijzigingen gecontroleerd werden doorgevoerd waren de Change Dependencies in de source base bekend.

Daarnaast zijn er verschillende gesprekken geweest om de juistheid en bruikbaarheid van de LCA ter discussie te stellen. Hierdoor zijn er verschillende tekortkomingen (***Tabel 5-6***) en bugs (***Tabel 5-7***) naar voren gekomen die vervolgens verholpen zijn.

Tekortkoming:
Code Churn Ratio filter ontbreekt.
Ook de verschillende services in de graaf tonen die wel gewijzigd zijn maar niet in een Change Dependency voorkomen.

Tabel 5-6, Tekortkomingen LCA

Bugs
Er werden Changesets getoond bij verschillende Workitems die in werkelijkheid niet aan elkaar gelinkt waren.
De Code Churn per service werd niet op de juiste manier bepaald.
Onder een service werden ook bestanden getoond die gewijzigd waren maar die deel uit maken van een andere service.

Tabel 5-7, Bugs LCA

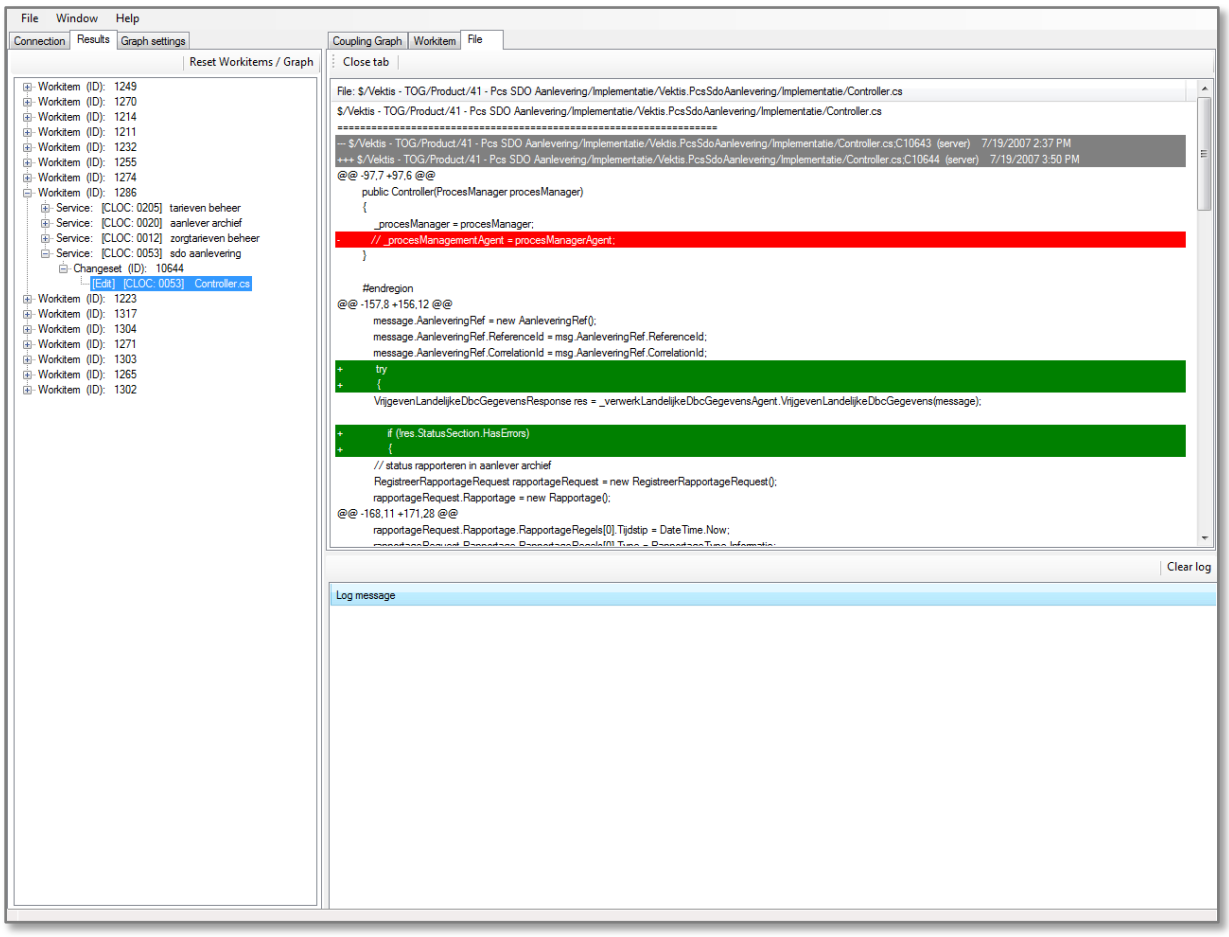
Nadat de gevonden tekortkomingen en bugs verholpen waren, toonde de LCA zonder problemen de Change Dependencies die zich in de Source base bevonden.

Om harde uitspraken te doen over de verschillende opgestelde toetsing criteria is moeilijk gebleken. Wanneer we kijken naar de Precission dan is deze 100% wanneer we uitgaan van het aantonen van Change Dependencies. Echter wanneer we kijken naar Logical Couplings is dit sterk

afhankelijk van hoe breed een bepaald Workitem omschreven is en hoe netjes de verschillende Changesets gelinkt zijn aan een Workitem (False-positives). Dit is per source base verschillend waardoor er geen harde uitspraak gedaan kan worden over de verschillende criteria. Daarnaast is het zo dat zolang een Logical Coupling niet gewijzigd is en dus niet voor komt in een change Dependency deze niet aangetoond wordt (false negative).

5.3 | Logical Couplings afleiden

Zoals al eerder besproken kan een Change Dependency duiden op een Logical Coupling. De LCA is zo opgebouwd dat het de gebruiker in staat stelt om een Change Dependency te onderzoeken of deze ook daadwerkelijk een Logical Coupling aantoont. Voor elke Change Dependency wordt de mogelijkheid geboden om de bijbehorende Workitems verder te onderzoeken. Door de file diff (***Figuur 5-11***) van de gewijzigde bestanden van een service te analyseren kan onderzocht worden of de Change Dependency een Logical Coupling aantoont.



Figuur 5-11, LCA toont file diff

6. | Resultaten

In dit hoofdstuk worden de resultaten voortkomend uit het onderzoek gepresenteerd. In hoofdstuk ***7| Evaluatie & Conclusies & Aanbevelingen*** zullen de resultaten besproken en geëvalueerd worden.

6.1 | Extracties Vektis

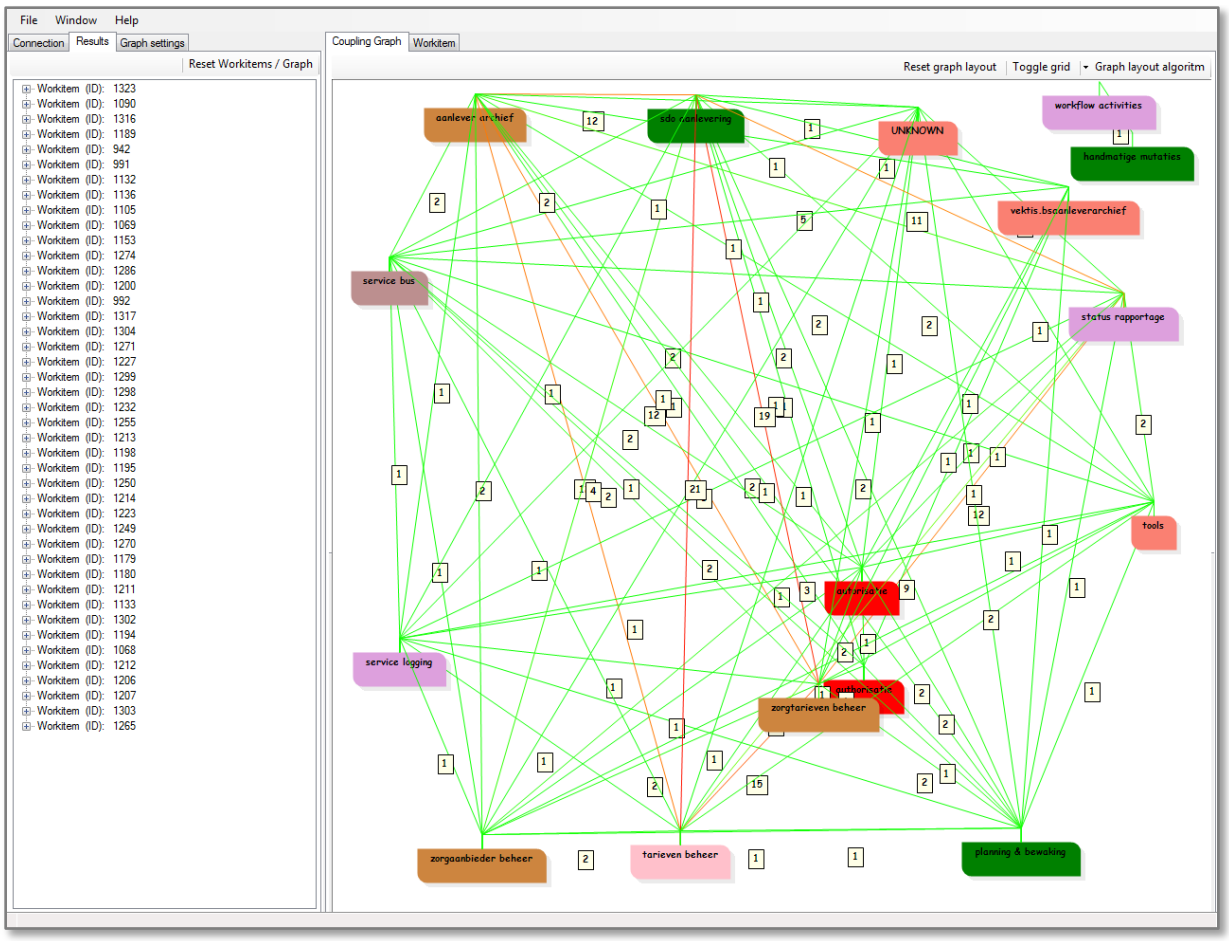
Om aan te tonen dat de methode ook daadwerkelijk een meerwaarde kan leveren op de kwaliteits tooling binnen de ontwikkelstraat Endeavour is de methode toegepast binnen een real life case. Voor de real life case is gekozen voor het project; Vektis – Tarieven Organen Gezondheidszorg. Binnen dit project, dat ongeveer 4 maanden in ontwikkeling was, is gebruik gemaakt van Workitems.

Er zijn verschillende extracties gedaan op de source base van Vektis elke extractie heeft andere filter instellingen. De verschillende filter instellingen worden hieronder (***Tabel 6-1***) beschreven.

Extractie:	Filter:	
EX1	Supported files:	C#
	Change type:	Edit
	Workitem type:	All
	Snapshot:	All Workitems
EX2	Supported files:	C#
	Change type:	Edit
	Workitem type:	Defect
	Snapshot:	All Workitems
EX3	Supported files:	C#
	Change type:	Edit
	Workitem type:	Change Request
	Snapshot:	All Workitems

Tabel 6-1, Filter instellingen extracties Vektis

Al snel bleek dat extractie EX1 te veel ruis (***Figuur 6-1***) vertoonde om een gedegen analyse op uit te voeren. Omdat in deze extractie ook bijvoorbeeld het Workitem type ‘Task’ is meegenomen ontstaan er veel False-positives. Tasks hebben doorgaans effect op het gehele systeem.



Figuur 6-1, LCA toont ruis na extractie

Uiteindelijk is ervoor gekozen om naar de extracties EX2 en EX3 te kijken. De verschillende Extracties zijn uitgewerkt terug te vinden in de bijlagen. (***Bijlage B***, ***Bijlage C***, ***Bijlage D***, ***Bijlage E***)

De verschillende extracties zijn doorlopen met leden van het ontwikkelteam van Vektis. Hieruit bleek dat de resultaten veel False Positives bevatten. Uiteindelijk konden er een aantal Logical Couplings worden aangetoond. De verschillende Logical Couplings die zijn aangetoond worden hieronder toegelicht.

6.1.1 | Infrastructure Component

Tussen verschillende CI's en het Infrastructure Component (IC) 'Status rapportage' werd een Change Dependency getoond. Na verdere analyse van de verschillende bijbehorende Workitems, Changesets en de gewijzigde bestanden bleek het te gaan om een Logical Coupling.

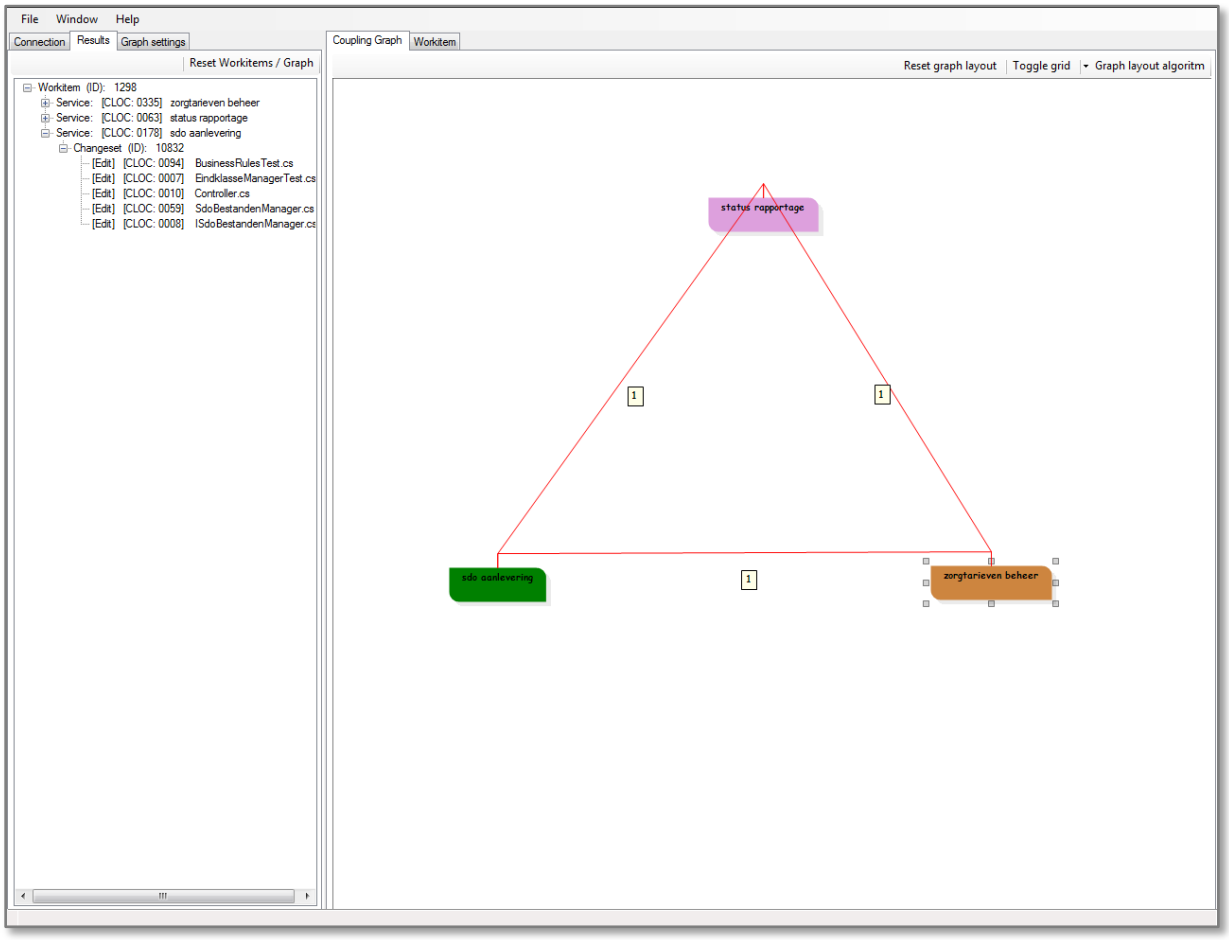
De uitwerkingen van deze resultaten zijn terug te vinden in Bijlage E.

Deze koppeling komt voort uit het feit dat het IC ‘Status rapportage’ deel uitmaakt van verschillende services. In dit geval werden bepaalde foutmeldingen centraal beheerd door het IC maar gebruikt door verschillende services. Wanneer bepaalde business rules geïmplementeerd worden of gewijzigd dienen hier eventueel ook wijzigingen aangebracht te worden.

Wanneer we kijken naar het feit dat de business rules binnen Vektis voornamelijk binnen de volgende twee services voorkomen:

- SDO Aanlevering
- Zorgtarieven beheer

Dan komt dit overeen met de Change Dependencies en de daaruit volgende Logical Coupling tussen deze services en de IC ‘Status rapportage’.



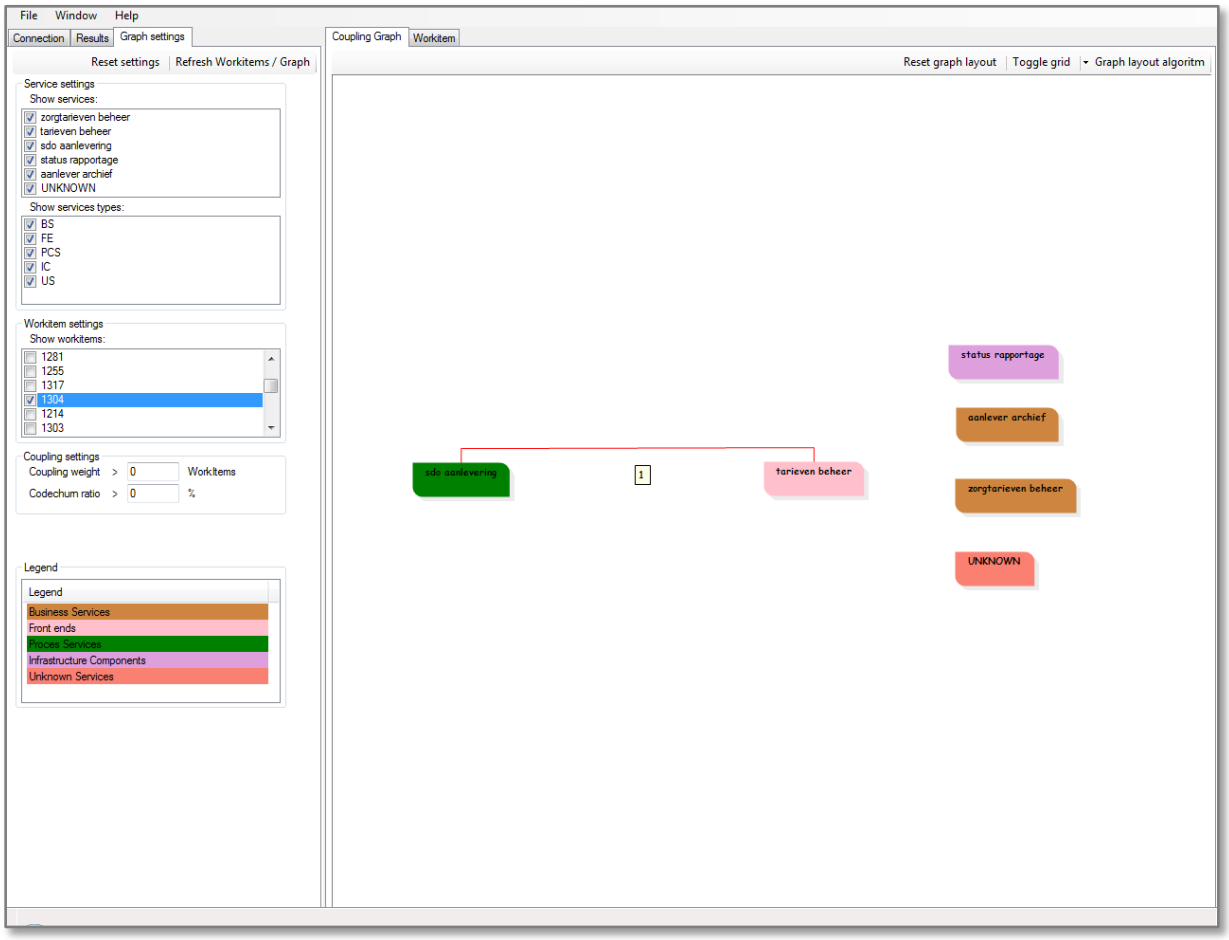
Figuur 6-2, LCA toont Change Dependency Infrastructure Component

6.1.2 | Front end

Tussen de service 'Tarieven beheer' en 'SDO aanlevering' werd ook een Change Dependency aangetoond (***Figuur 6-3***) en na verdere analyse bleek ook hier een Logical Coupling aanwezig te zijn.

De uitwerkingen van deze resultaten zijn terug te vinden in Bijlage D.

Deze koppeling was gebaseerd op het feit dat er een bug bestond in het proces welke vastgelegd is in de process service 'SDO aanlevering'. Het proces wordt echter aangestuurd vanuit de Front end 'Tarieven beheer' die dus ook gewijzigd moest worden.



Figuur 6-3, LCA toont Change Dependency Front end

6.2 | Naleving richtlijnen Endeavour

Tijdens het verzamelen van de resultaten werd al snel het belang van het zorgvuldig naleven van de gestelde richtlijnen binnen Endeavour duidelijk. Zoals in de vorige paragraaf wordt beschreven waren er zeer veel False-positives aanwezig in de resultaten. De aanname ***Aanname 4-1*** was helaas niet altijd van toepassing doordat er af en toe wijzigigingen hadden plaats gevonden onder een Workitem die niet gerelateerd waren aan de omschrijving van het Workitem. Daarnaast was de aanname ***Aanname 5-1***

helaas ook niet altijd terecht en waren er verschillende Changesets die aan verschillende Workitems gekoppeld waren.

Door het uitvoeren van verschillende analyses op de verschillende extracties kunnen dergelijke overtredingen van de richtlijnen geconstateerd worden. Alhoewel het onderzoek niet als doel had om de werkwijze van de ontwikkelaars te analyseren is gebleken dat als bijkomend voordeel van de LCA dit wel mogelijk is. De richtlijnen zijn net als de rest van Endeavour opgesteld om de productiviteit en effectiviteit van software ontwikkeling op een gunstige manier te beinvloeden en te controleren. De ontwikkelde LCA zou een toezicht houdende rol in dit doel kunnen vervullen.

7. | Evaluatie & Conclusies & Aanbevelingen

In dit hoofdstuk wordt het onderzoek geëvalueerd en de getrokken conclusies besproken. Daarnaast worden verschillende aanbevelingen gedaan voor richtingen waarin het onderzoek uitgebreid en/of aangepast zou kunnen worden.

7.1 | Evaluatie

Wanneer we naar de resultaten kijken, worden er veel False Positives in de extracties van Vektis aangetoond. Dit heeft verschillende oorzaken die hieronder worden beschreven.

Oorzaken
Workitems zijn te breed omschreven. Hierdoor vallen verschillende wijzigingen onder een Workitem waardoor False Positives ontstaan in de Change Dependencies.
Changesets dienen handmatig aan een Workitem gekoppeld te worden. Dit was helaas niet altijd even correct gebeurd. Hierdoor kwamen de Changesets niet altijd overeen met de Workitem omschrijving.
Sommige wijzigingen waren doorgevoerd zonder dat ze opgenomen waren in een Workitem. Vervolgens maakte deze wijziging wel deel uit van een Changeset die aan een Workitem gelinkt was.

Tabel 7-1, Oorzaken False Positives

Om een beter beeld te krijgen van de effectiviteit van de LCA zouden verschillende precondities opgesteld moeten worden voor de inrichting van de source base. De precisie van de LCA is namelijk afhankelijk van deze inrichting en de manier waarop deze verder gevuld wordt. False positives worden uiteindelijk niet door de LCA gecreëerd maar door de manier waarop de verschillende Changesets aan Workitems zijn gelinkt en hoe scherp de Workitems gedefinieerd zijn.

Een aanbeveling voor de inrichting van de Workitems zou zijn om deze betrekking te laten hebben op een service, proces of een business rule. Hierdoor zullen Change Dependencies meer Logical Couplings aantonen en minder False-positives.

7.2 | Conclusies

Het Onderzoek levert uiteindelijk één conclusie op, deze wordt hieronder beschreven.

LCA levert meerwaarde

Op basis van de resultaten kan geconcludeerd worden dat de toegepaste methode een meerwaarde kan leveren op de bestaande kwaliteitstooling. De hypothese ***Hypothese 4-1*** gesteld aan het begin van het

onderzoek is juist gebleken: er kunnen aan de hand van Change Dependencies, Logical Couplings worden aangetoond. Dit maakt het mogelijk om de status van de architectuur te monitoren.

Als bijkomend voordeel heeft het onderzoek nog een ander resultaat opgeleverd. De LCA maakt het mogelijk om toezicht te houden of de opgelegde werkwijze binnen Endavour wordt nageleefd. Wanneer er niet op de juiste manier wordt omgegaan met het uitvoeren van onderhoud, door de administratie van de Workitems en Changesets niet op de voorgeschreven manier uit te voeren, zal de LCA ongewenste resultaten (false-positives) genereren. Dit tezamen met het aantonen van Logical Couplings ondersteunt het doel van de ontwikkelstraat het verbeteren van de productiviteit en effectiviteit bij het ontwikkelen van software.

7.3 | Toekomstig werk

Het beschreven onderzoek richt zich in eerste instantie alleen op het aantoonbaar en inzichtelijk maken van de wijzigingen binnen een service en de verschillende Logical Couplings, tussen services, die tijdens de ontwikkeling van een systeem, met een SOA, ontstaan. Code Churn vormt de basis om change sets te kunnen bepalen echter in de toekomst zou er nog aanvullend onderzoek verricht kunnen worden. Hieronder worden nog andere richtingen beschreven.

Precondities

Om de methode een meerwaarde te kunnen laten leveren dient de inrichting van de Workitems binnen de source base aan bepaalde precondities te voldoen. In het onderzoek is gebleken dat het mogelijk is om een Workitem precies of juist breed te specificeren wat kan leiden tot false-positives. In de toekomst dient er onderzoek gedaan te worden naar de meest ideale specificatie.

Code Churn als filter voor Change Dependencies

In het huidige onderzoek is voor de automatische analyse alleen gebruik gemaakt van de wijzigingsgeschiedenis. Door ook de wijzigingen te bekijken en hier bepaalde types in te herkennen kunnen de resultaten mogelijkerwijs nauwkeuriger gemaakt worden. Door gebruik te maken van de Abstract Syntax tree (AST) kunnen de verschillende entiteiten binnen de classe worden onderzocht.

Overeenkomsten in Code Churn binnen Change Dependencies

Het meten van de overeenkomsten tussen de Code Churn van de verschillende classes in dezelfde change set kan interessante uitkomsten bieden. Wanneer naar voren komt dat een Change Dependency voortkomt uit functionaliteit die op meer dan één plek staat beschreven (Bijvoorbeeld door een business rule), geeft dit direct inzicht in wat de oorzaak van de afhankelijkheid is.

Gedetailleerde views

In het huidige onderzoek is er voornamelijk gekeken naar de koppelingen die aangetoond konden worden tussen de verschillende services wat het mogelijk maakt om het principe van Loose Coupling te analyseren. Services zijn de grootste eenheden waarin de architectuur van de systemen ontwikkeld binnen Endeavour in opgedeeld kunnen worden. Door de services weer op te delen in componenten kan er onderzoek gedaan worden naar de koppelingen tussen de verschillende componenten binnen een service de zogenaamde Cohession. Maar ook de koppelingen die van een bepaalde component binnen een service naar een ander component binnen een andere service kunnen onderzocht worden.

Toevoegen van informatie aan Source Base

Wanneer men meer informatie zou kunnen toevoegen aan de source base kan men deze informatie meenemen in de analyse die uitgevoerd wordt door de LCA. Hierdoor kunnen er geautomatiseerd nauwkeurigere resultaten behaald worden. Denk aan bijvoorbeeld een Workitem waarin geregistreerd staat of de wijziging betrekking heeft op de contracten en de servicebus. Hierdoor zouden er extra filters mogelijk zijn op de uiteindelijke extractie.

8. | Verklarende Woordenlijst

In onderstaande verklarende woordenlijst vindt u uitleg over verschillende gebruikte termen of afkortingen.

Term / Afkorting:	Verklaring:
Abstract Syntax tree (AST)	Een tree representatie van de verschillende entiteiten in de source code van een classe
Autonoom	Zelfstandig, onafhankelijk
Configuration Item (CI)	Een service binnen een SOA volgens de richtlijnen van de ontwikkelstraat Endeavour
Entiteit	Iets dat wezenlijk bestaat
Heterogeen	Ongelijksoortig
Solution .Net	Verzameling van .Net projecten
Source base	Locatie waar source code opgeslagen wordt en versie beheer toegepast wordt
Source code	Broncode van een stuk software

9. | Bibliografie

AskBurton. (2004, 9 9). *Visual Studio Team System*. Opgeroepen op 8 22, 2007, van Visual Studio Team System: http://blogs.msdn.com/askburton/archive/2004/09/09/227515.aspx

Ball, T., & Nagappan, N. (2005). Use of Relative Code Churn Measures to Predict System Defect Density. *Proceedings of the 27th international conference on Software engineering* (pp. 284 - 292). New York: ACM Press.

D'Ambros, M., Lanza, M., & Lungu, M. (2006). The evolution radar: visualizing integrated logical coupling information. *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories.* New York: ACM Press.

de Vries, M. (2006). ENDEV01 - Logical Reference Architecture. Veenendaal: Infosupport.

de Vries, M. (2006). ENDEV02 - Technical Reference Architecture. Veenendaal: Infosupport.

de Weerd, E. (2006). Endeavour Introductie. Veenendaal: Infosupport.

Documentation of the Netron Project. (sd). Opgeroepen op 6 1, 2007, van Netron: http://netron-doc.kuss.dyndns.org/pmwiki/pmwiki.php/Tutorials/QuickStart

Fernström, C., Närfelt, K.-H., & Ohlsson, L. (1992). Software Factory Principles, Architecture, and Experiments. *9* (2).

Gall, H., Jazayeri, M., & Krajewski, J. (2003). CVS Release History Data for Detecting Logical Couplings. *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution.* Washington: IEEE Computer Society.

Hodges, B. (sd). *Team Foundation Version Control API documentation*. Opgeroepen op 5 1, 2007, van MSDN Blogs: http://blogs.msdn.com/buckh/archive/2005/12/09/502179.aspx

Huhns, M. N., & Singh, M. P. (2005). Service-Oriented Computing: Key Concepts and Principles. *9* (1).

Lines of code metrics. (sd). Opgehaald van Aivosto: http://www.aivosto.com/project/help/pm-loc.html

Munson, J. C., & Elbaum, S. G. (1998). Code Churn: A Measure for Estimating the Impact of Code Change. *ICSM '98: Proceedings of the International Conference on Software Maintenance.* Washington: IEEE Computer Society.

Relevance, Recall, Precision, Fallout. (2000). (The Regents of the University of California) Opgeroepen op 8 13, 2007, van Law Library — University of California, Berkeley: http://www.law.berkeley.edu/library/classes/alr/boolean.html

Smit, C. (2003). Why Service-Oriented Architecture. SystemicLogic© 2003 Research.

Snijder, J. (2007). *Measurement Construct - Code Churn.* Veenendaal: Info Support B.V.

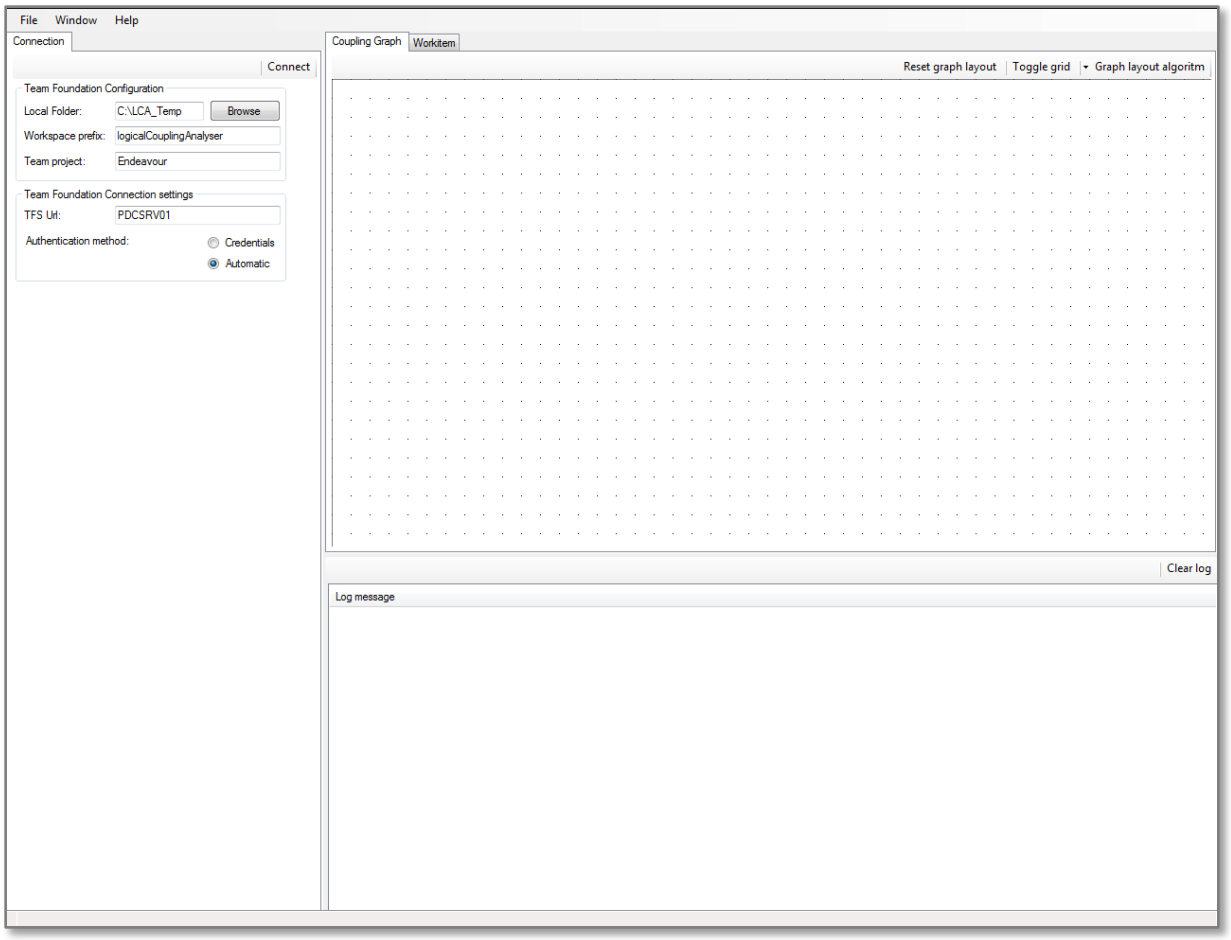
Statement counts. (sd). Opgehaald van Aivosto: http://www.aivosto.com/project/help/pm-stmt.html

Bijlage A

Logical Coupling Analyser (LCA)

Inleiding

De LCA (***Figuur 0-1***) is een PoC speciaal ontwikkeld voor de ontwikkelstraat Endeavour van Info Support. De LCA dient een aanvulling te vormen op de bestaande kwaliteitstooling binnen Endeavour. Aan de hand van de resultaten van de LCA kunnen bepaalde uitspraken gedaan worden over de status van de onderliggende architectuur van de te onderzoeken systemen.



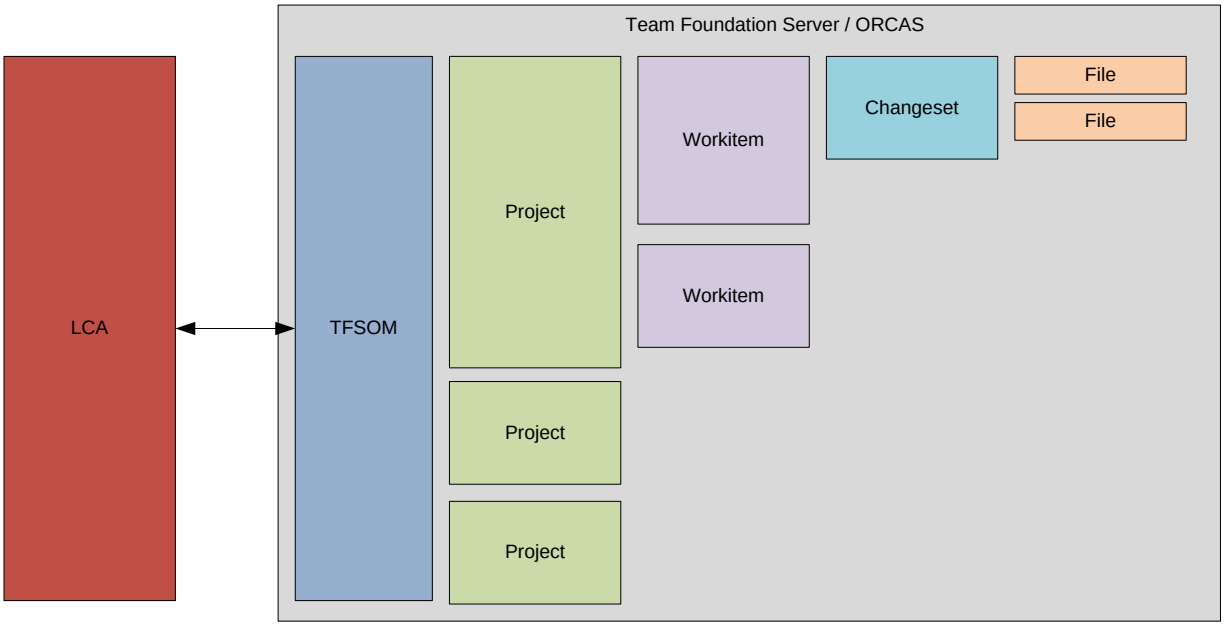
Figuur 0-1, Logical Coupling Analyser (LCA) start scherm

Endeavour

Endeavour is de ontwikkelstraat van Info Support. Binnen de ontwikkelstraat wordt een bepaalde architectuur voorgeschreven de zogenaamde Logische Referentie Architectuur (LRA) welke geïmplementeerd dient te worden door de te ontwikkelen systemen. Aan deze LRA zijn verschillende regels verbonden over hoe bepaalde koppelingen aanwezig mogen zijn. De LCA dient het mogelijk te maken de verschillende Logical Couplings binnen het systeem inzichtelijk te maken.

Teamfoundation Server

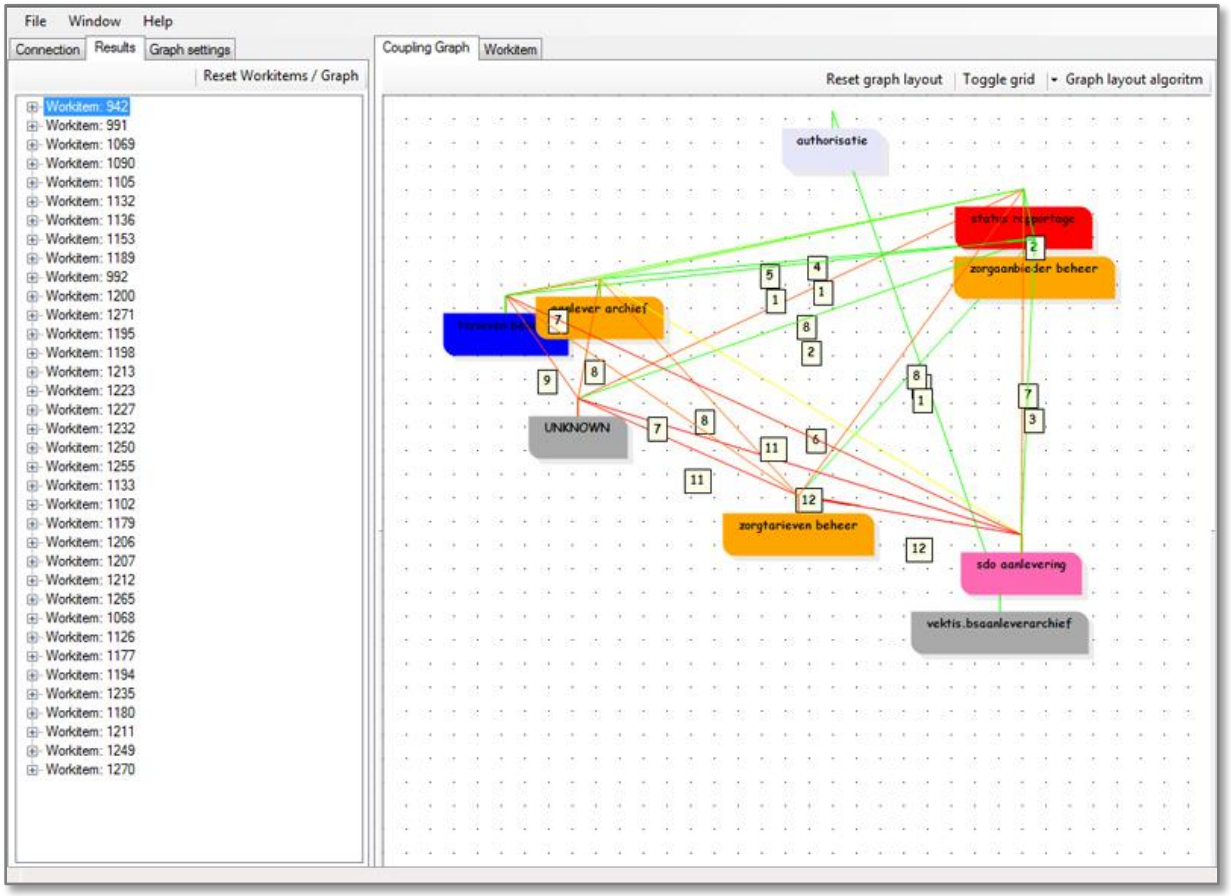
De LCA is ontwikkeld om systemen te analyseren die opgeslagen zijn in een Team Foundation Server of zijn opvolger 'Orcas'. De LCA maakt gebruik van het object model van Team Foundation Server (TFSOM) om informatie uit te wisselen. Met behulp van het TFSOM worden de Workitems van het te analyseren Team project opgevraagd en geanalyseerd (***Figuur 0-2***).



Figuur 0-2, Voorstelling LCA tegenover TFSOM

Netron Diagramming Library

In de LCA wordt gebruik gemaakt van de Netron Diagramming Library[7] om de resultaten in een graaf (***Figuur 0-3***) weer te kunnen geven. Dit maakt het mogelijk om de resultaten visueel inzichtelijk te maken. Men kan vervolgens de resultaten filteren en op koppelingen klikken om deze verder te analyseren



Figuur 0-3, Voorbeeld van een graaf gevisualiseerd binnen de LCA

De library is een open source kit om snel en gemakkelijk bepaalde data te visualiseren. De kit is onder de volgende licentie te gebruiken:

De Netron Diagramming Library heeft een licentie van het type: *GPL (General Public License[8])*

Punt van aandacht 1, Netron Diagramming Library licentie type

[7] Voor meer informatie over de Netron Diagramming Library: http://www.orbifold.net/Netron/info.php
[8] Voor meer informatie over GPL licentie: http://www.gnu.org/copyleft/gpl.html

Bijlage B

Extractie EX2

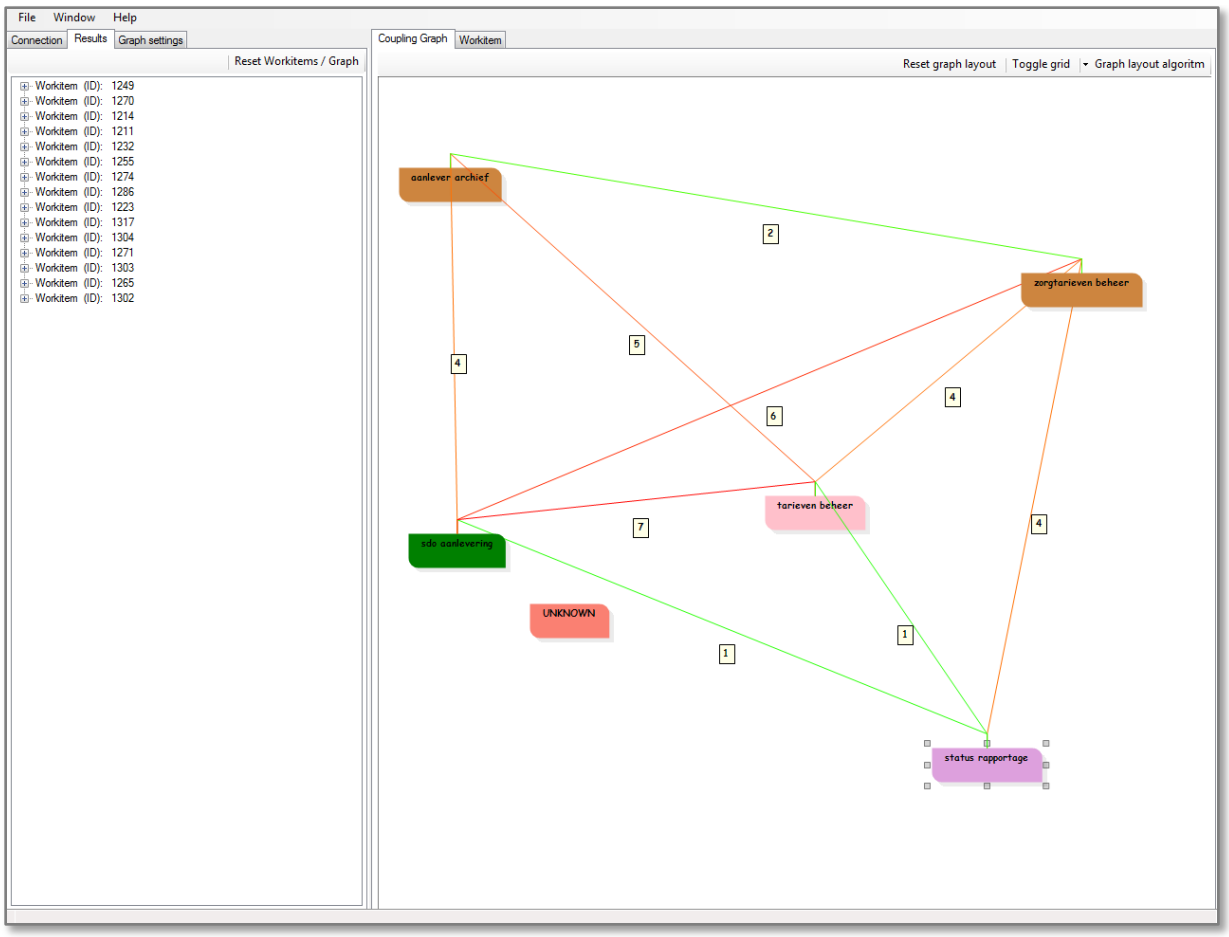
Filter instellingen voor extractie:

Filter:	Instelling:
Supported files	C#
Change type	Edit
Workitem type	Defect
Snapshot:	All Workitems

Change Dependencies:

Indicator	Coupling weight	Service x	Service y
CD01	1	FE - tarieven beheer	IC - status rapportage
CD02	1	PCs - sdo aanlevering	IC - status rapportage
CD03	2	BS - aanlever archief	BS - zorgtarieven beheer
CD04	4	IC - status rapportage	BS - zorgtarieven beheer
CD05	4	BS - aanlever archief	PCs - sdo aanlevering
CD06	4	FE - tarieven beheer	BS - zorgtarieven beheer
CD07	5	BS - aanlever archief	FE - tarieven beheer
CD08	6	BS - zorgtarieven beheer	PCs - sdo aanlevering
CD09	7	FE - tarieven beheer	PCs - sdo aanlevering

Graaf:



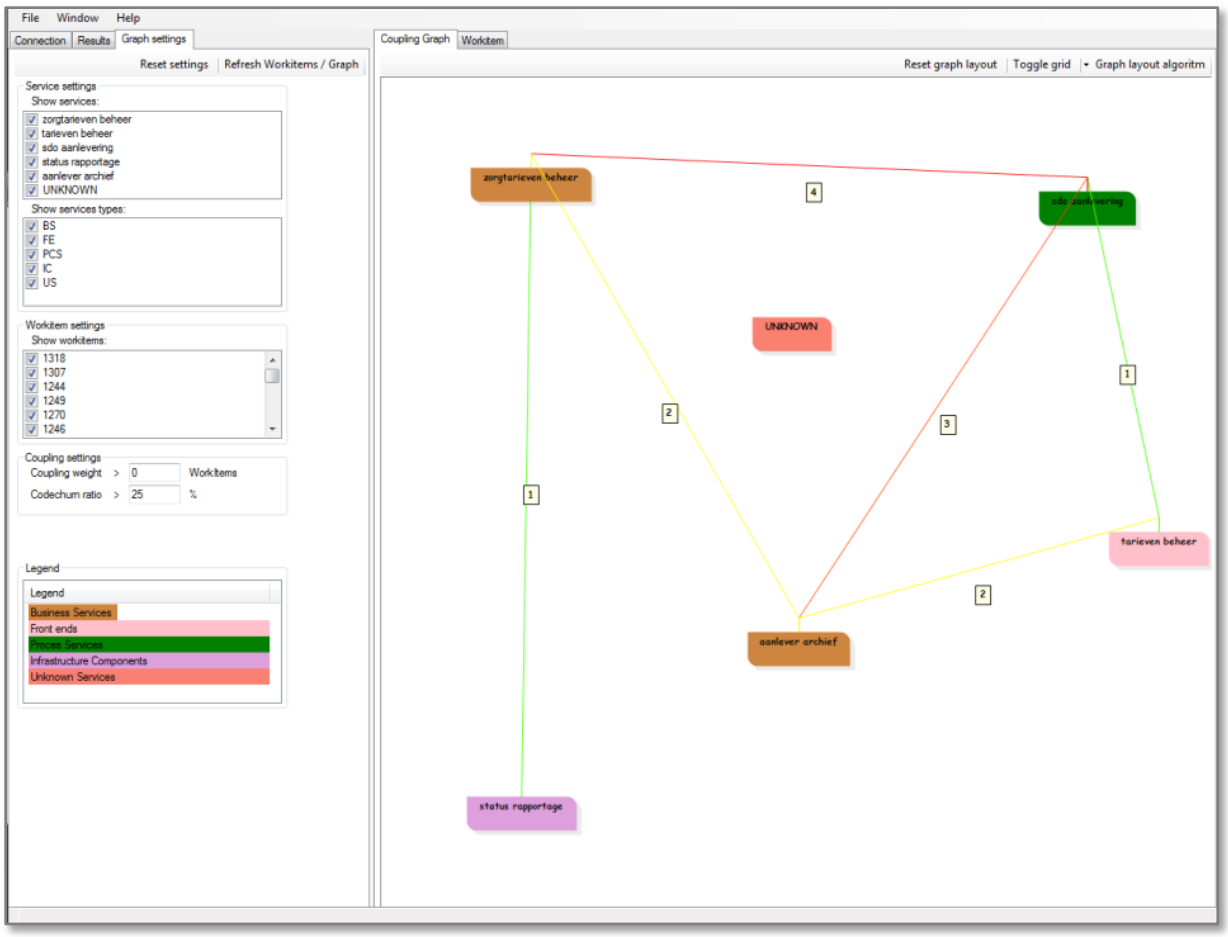
Filter instellingen na extractie:

Filter:	Instelling:
Code Churn Ratio	25 %

Change Dependencies:

Indicator	Coupling weight	Service x	Service y
CD03	2	BS - aanlever archief	BS - zorgtarieven beheer
CD04	1	IC - status rapportage	BS - zorgtarieven beheer
CD05	3	BS - aanlever archief	PCs - sdo aanlevering
CD07	2	BS - aanlever archief	FE - tarieven beheer
CD08	4	BS - zorgtarieven beheer	PCs - sdo aanlevering
CD09	1	FE - tarieven beheer	PCs - sdo aanlevering

Graaf:


File
Window
Help
Connection
Results
Graph settings
Reset settings
Refresh Workitems / Graph
Service settings
Show services:
zorgtarieven beheer
tarieven beheer
sdo aanlevering
status rapportage
aanlever archief
UNKNOWN
Show services types:
BS
FE
PCS
IC
US
Workitem settings
Show workitems:
1318
1307
1244
1249
1270
1246
Coupling settings
Coupling weight > 0 Workitems
Codechum ratio > 25 %
Legend
Legend
Business Services
Front ends
Proces Services
Infrastructure Components
Unknown Services
Coupling Graph
Workitem
Reset graph layout
Toggle grid
Graph layout algoritm
zorgtarieven beheer
sdo aanlevering
UNKNOWN
tarieven beheer
aanlever archief
status rapportage
4
1
2
3
1
2

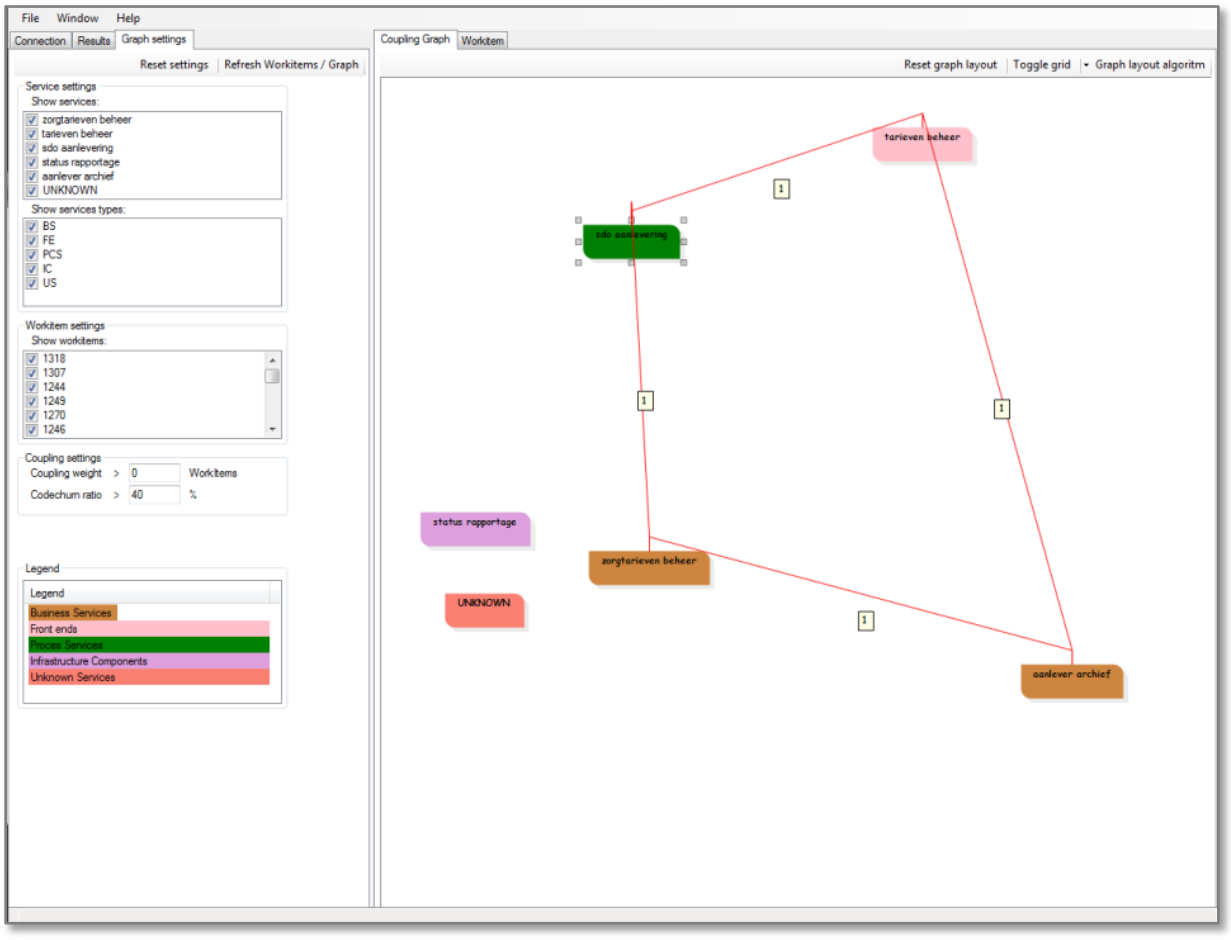
Filter instellingen na extractie:

Filter:	Instelling:
Code Churn Ratio	45 %

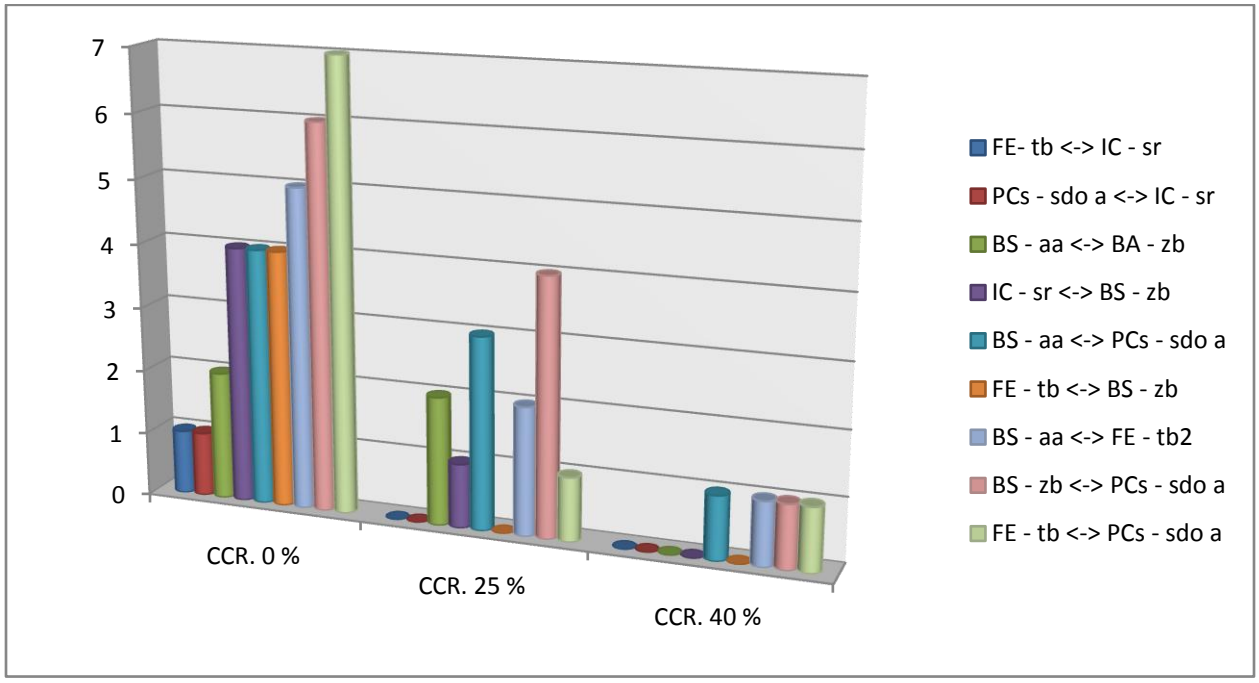
Change Dependencies:

Indicator	Coupling weight	Service x	Service y
CD05	1	BS - aanlever archief	PCs - sdo aanlevering
CD07	1	BS - aanlever archief	FE - tarieven beheer
CD08	1	BS - zorgtarieven beheer	PCs - sdo aanlevering
CD09	1	FE - tarieven beheer	PCs - sdo aanlevering

Graaf:



Verloop:


7
6
5
4
3
2
1
0
CCR. 0 %
CCR. 25 %
CCR. 40 %
FE- tb <-> IC - sr
PCs - sdo a <-> IC - sr
BS - aa <-> BA - zb
IC - sr <-> BS - zb
BS - aa <-> PCs - sdo a
FE - tb <-> BS - zb
BS - aa <-> FE - tb2
BS - zb <-> PCs - sdo a
FE - tb <-> PCs - sdo a

Bijlage C

Extractie EX3

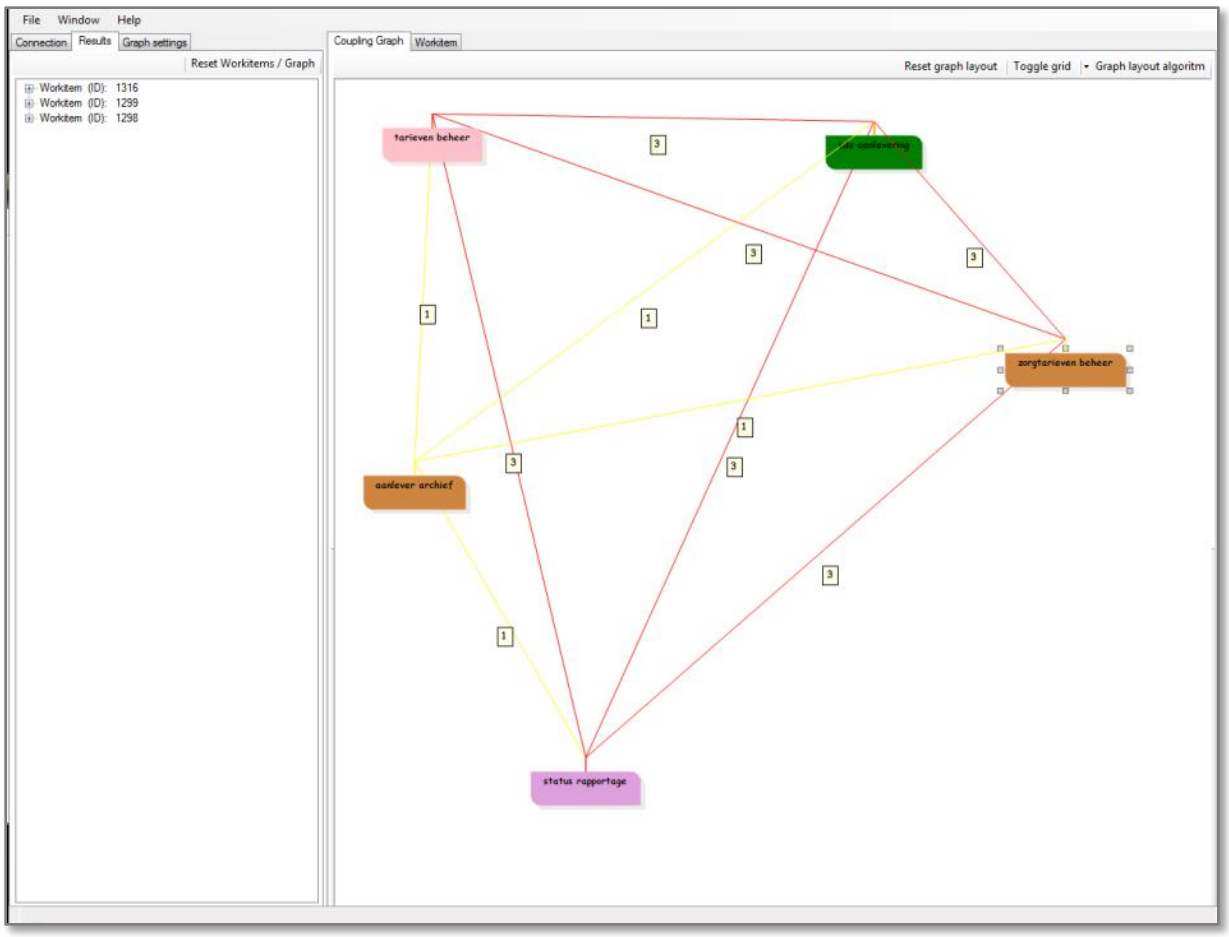
Filter instellingen voor extractie:

Filter:	Instelling:
Supported files	C#
Change type	Edit
Workitem type	Change Request
Snapshot:	All Workitems

Change Dependencies:

Indicator	Coupling weight	Service x	Service y
CD01	3	FE - tarieven beheer	IC - status rapportage
CD02	3	PCs - sdo aanlevering	IC - status rapportage
CD03	3	BS - zorgtarieven beheer	IC - status rapportage
CD04	1	BS - aanlever archief	IC - status rapportage
CD05	1	BS - aanlever archief	PCs - sdo aanlevering
CD06	1	BS - aanlever archief	FE - tarieven beheer
CD07	1	BS - aanlever archief	BS - zorgtarieven beheer
CD08	3	BS - zorgtarieven beheer	PCs - sdo aanlevering
CD09	3	BS - zorgtarieven beheer	FE - tarieven beheer
CD10	3	PCs - sdo aanlevering	FE - tarieven beheer

Graaf:



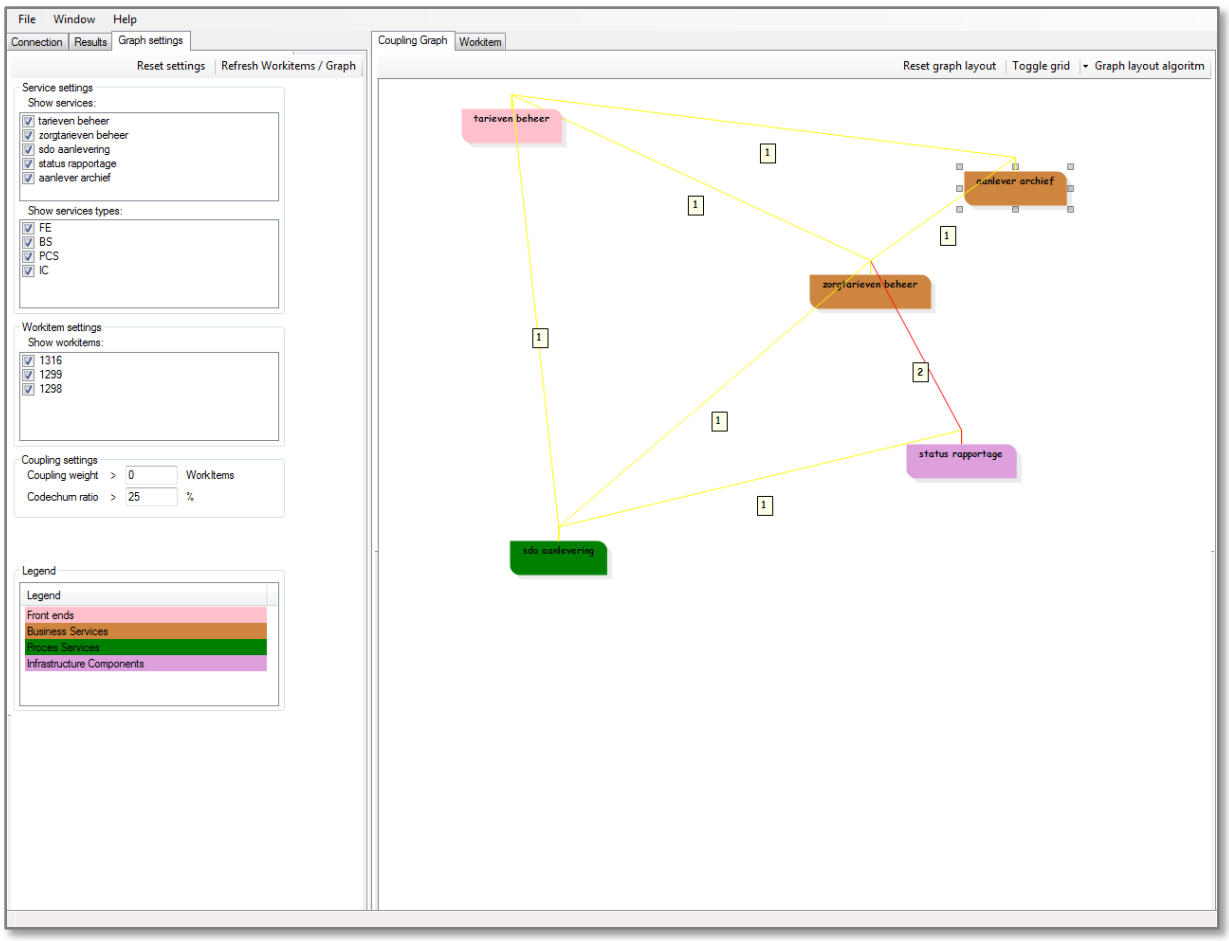
Filter instellingen na extractie:

Filter:	Instelling:
Code Churn Ratio	25 %

Change Dependencies:

Indicator	Coupling weight	Service x	Service y
CD02	1	PCs - sdo aanlevering	IC - status rapportage
CD03	2	BS - zorgtarieven beheer	IC - status rapportage
CD06	1	BS - aanlever archief	FE - tarieven beheer
CD07	1	BS - aanlever archief	BS - zorgtarieven beheer
CD08	1	BS - zorgtarieven beheer	PCs - sdo aanlevering
CD09	1	BS - zorgtarieven beheer	FE - tarieven beheer
CD10	1	PCs - sdo aanlevering	FE - tarieven beheer

Graaf:


File
Window
Help
Connection
Results
Graph settings
Reset settings
Refresh Workitems / Graph
Service settings
Show services:
tarieven beheer
zorgtarieven beheer
sdo aanlevering
status rapportage
aanlever archief
Show services types:
FE
BS
PCS
IC
Workitem settings
Show workitems:
1316
1299
1298
Coupling settings
Coupling weight > 0 WorkItems
Codechum ratio > 25 %
Legend
Legend
Front ends
Business Services
Proces Services
Infrastructure Components
Coupling Graph
Workitem
Reset graph layout
Toggle grid
Graph layout algoritm
tarieven beheer
aanlever archief
zorgtarieven beheer
status rapportage
sdo aanlevering
1
1
1
1
2
1
1

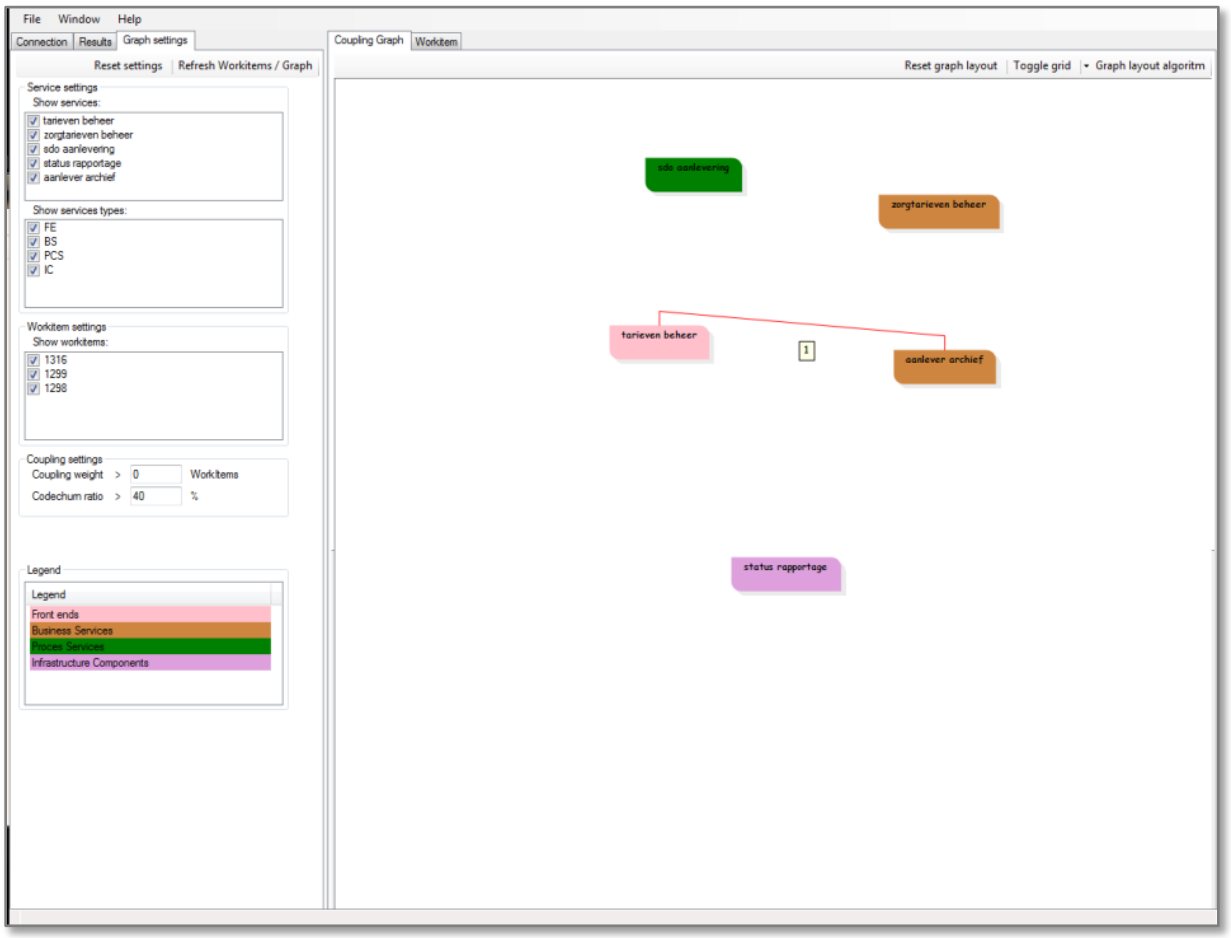
Filter instellingen na extractie:

Filter:	Instelling:
Code Churn Ratio	40 %

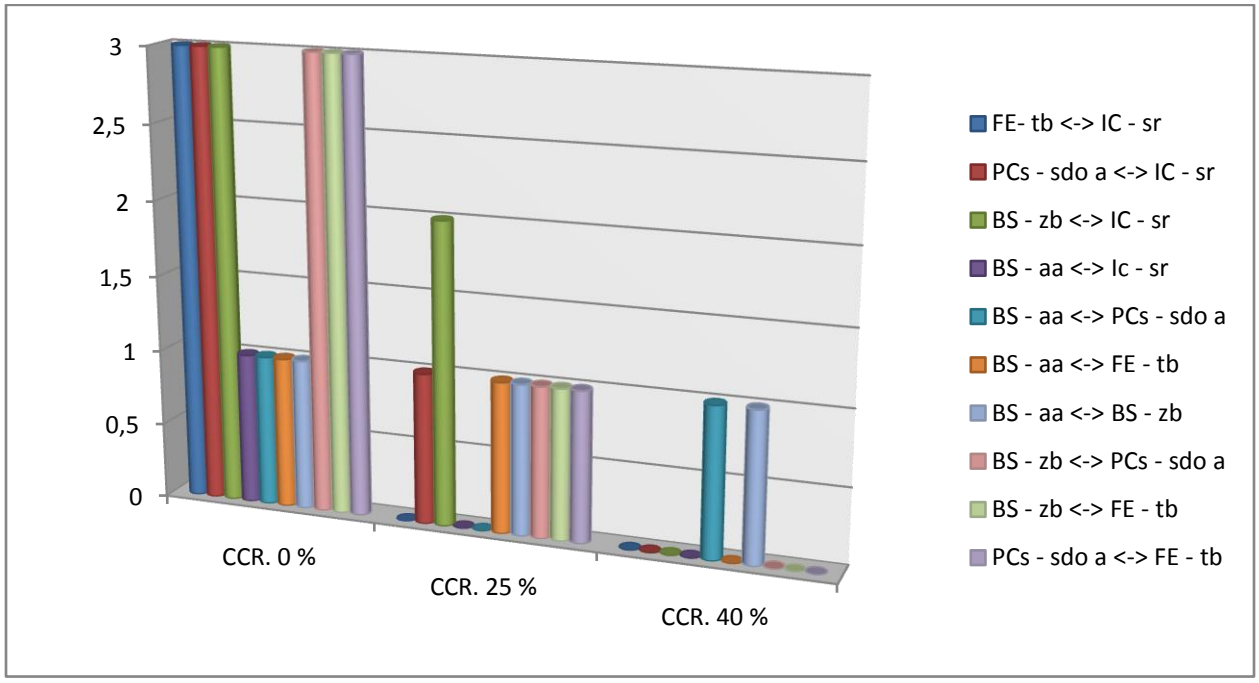
Change Dependencies:

Indicator	Coupling weight	Service x	Service y
CD07	1	BS - aanlever archief	FE - tarieven beheer

Graaf:



Verloop:


3
2,5
2
1,5
1
0,5
0
CCR. 0 %
CCR. 25 %
CCR. 40 %
FE- tb <-> IC - sr
PCs - sdo a <-> IC - sr
BS - zb <-> IC - sr
BS - aa <-> Ic - sr
BS - aa <-> PCs - sdo a
BS - aa <-> FE - tb
BS - aa <-> BS - zb
BS - zb <-> PCs - sdo a
BS - zb <-> FE - tb
PCs - sdo a <-> FE - tb

Bijlage D
Logical Coupling Front end

Filter

Om de Logical Coupling aan te tonen tussen FE - tarieven beheer en PCS – SDO aanlevering zijn de volgende filterinstellingen toegepast op Extractie EX2 (***Bijlage B***).

Filter:	Instelling:
Show Workitem	1304

Tabel 0-1, Filter instellingen

Hierdoor worden alleen zichtbaar de Change Dependencies die voor Workitem 1304 gelden.

1304	**Front end toont altijd rapport van eerste proces**	**Change Request**
Telkens wordt de rapportage van de eerste aanlevering getoond als je vanuit proces overzicht doorklikt. *Update 24-07: het probleem bleek te zitten in de proces service: bij iedere workflow instance werden de activities van alle workflows toegevoegd, met als gevolg dat de juiste status niet (altijd) werd bepaald.*		

Tabel 0-2, Workitem 1304

Change Dependencies

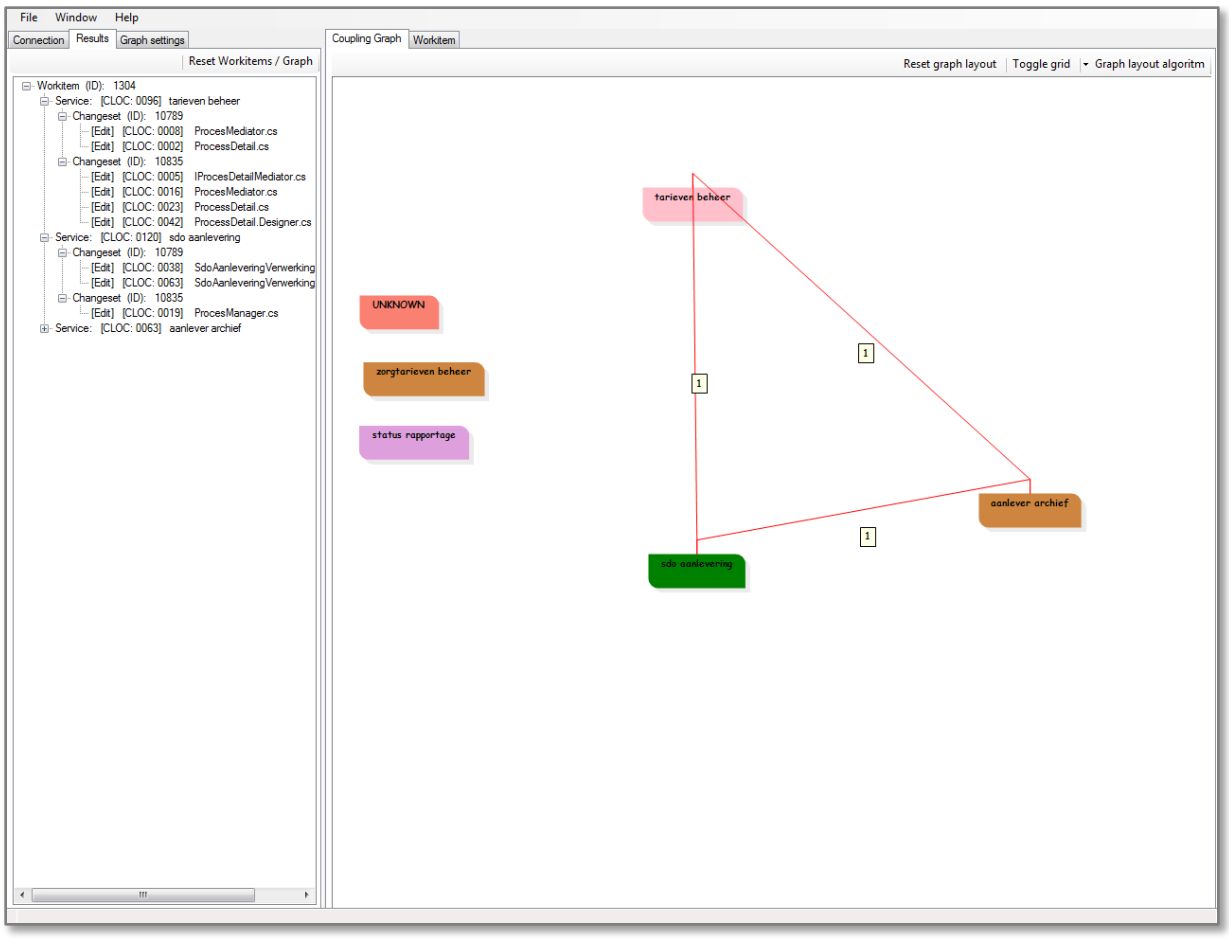
Na het toepassen van de filter bleven de volgende Change Dependencies zichtbaar.

Indicator	Coupling weight	Service x	Service y
CD05	1	BS - aanlever archief	PCs - sdo aanlevering
CD07	1	BS - aanlever archief	FE - tarieven beheer
CD09	1	FE - tarieven beheer	PCs - sdo aanlevering

Tabel 0-3, Change Dependencies

Graaf

In de graaf zijn de Change Dependencies die voortkomen uit Workitem 1304 zichtbaar.

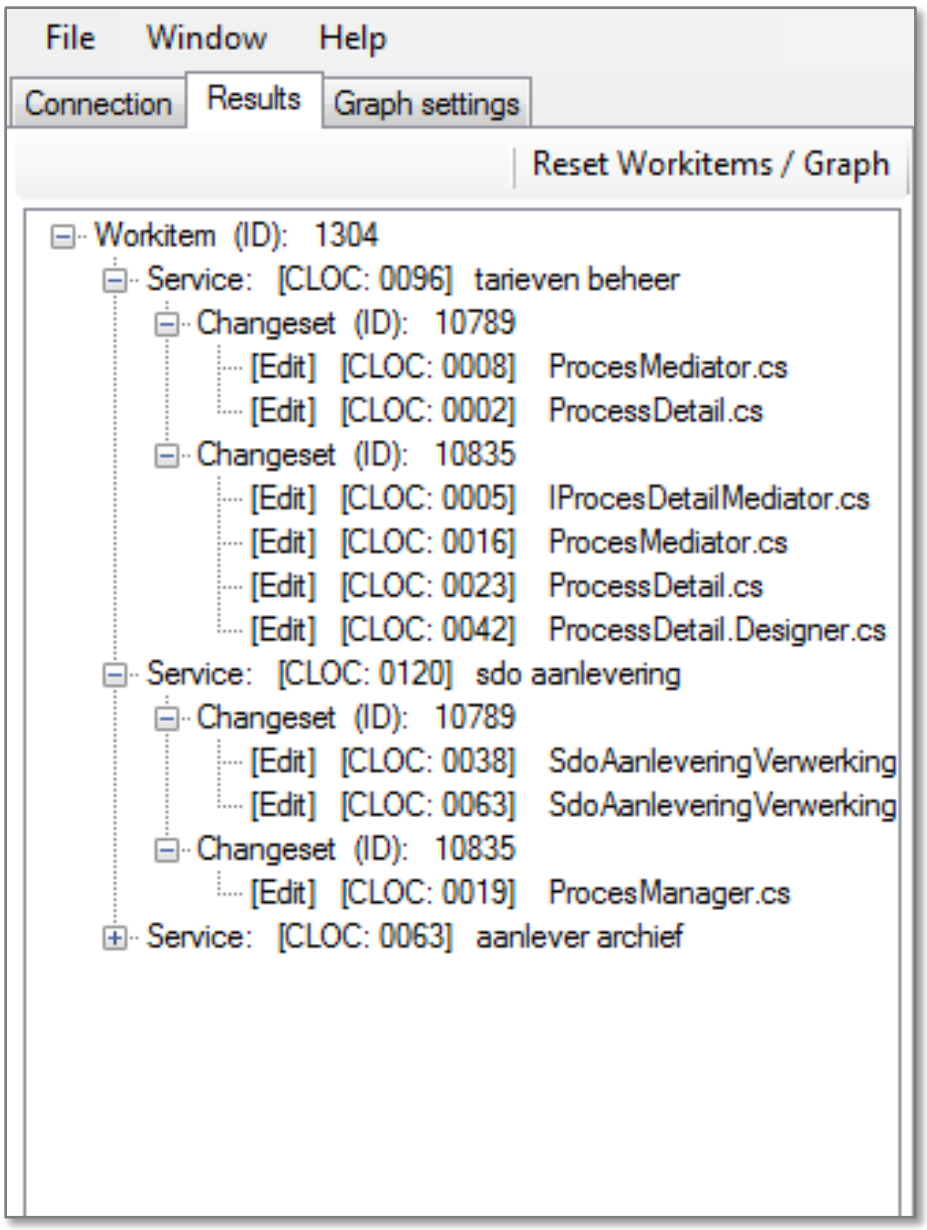


Figuur 0-1, LCA toont Change Dependency graaf

Workitems

Wanneer we kijken naar de Workitems blijken er twee Changesets aan het Workitem 1304 gelinkt te zijn:

- Changeset 10789
- Changeset 10835



Figuur 0-2, Results tab toont Workitems

Bijlage E
Logical Coupling Infrastructural Component

Filter

Om de Logical Coupling aan te tonen tussen BS - zorgtarieven beheer en IC - status rapportage zijn de volgende filterinstellingen toegepast op Extractie EX3 (***Bijlage C***).

Filter:	Instelling:
Show Workitem	1298

Tabel 0-1, Filter instellingen

Hierdoor worden alleen zichtbaar de Change Dependencies die voor Workitem 1298 gelden.

1298	Zorgvraag code terugbrengen naar 2 posities	Change Request
De zorgvraag code is soms 3 posities lang. Deze moet worden teruggebracht naar 2 posities tijdens het inlezen in het nieuwe systeem? of alleen in TOG-inp en bij DR0012? *"061" -> "61" (voorloopnul weg)* *"160" -> "16" (trailing nul weg)* *"123" -> alarm!*		

Tabel 0-2, Workitem 1298

Change Dependencies

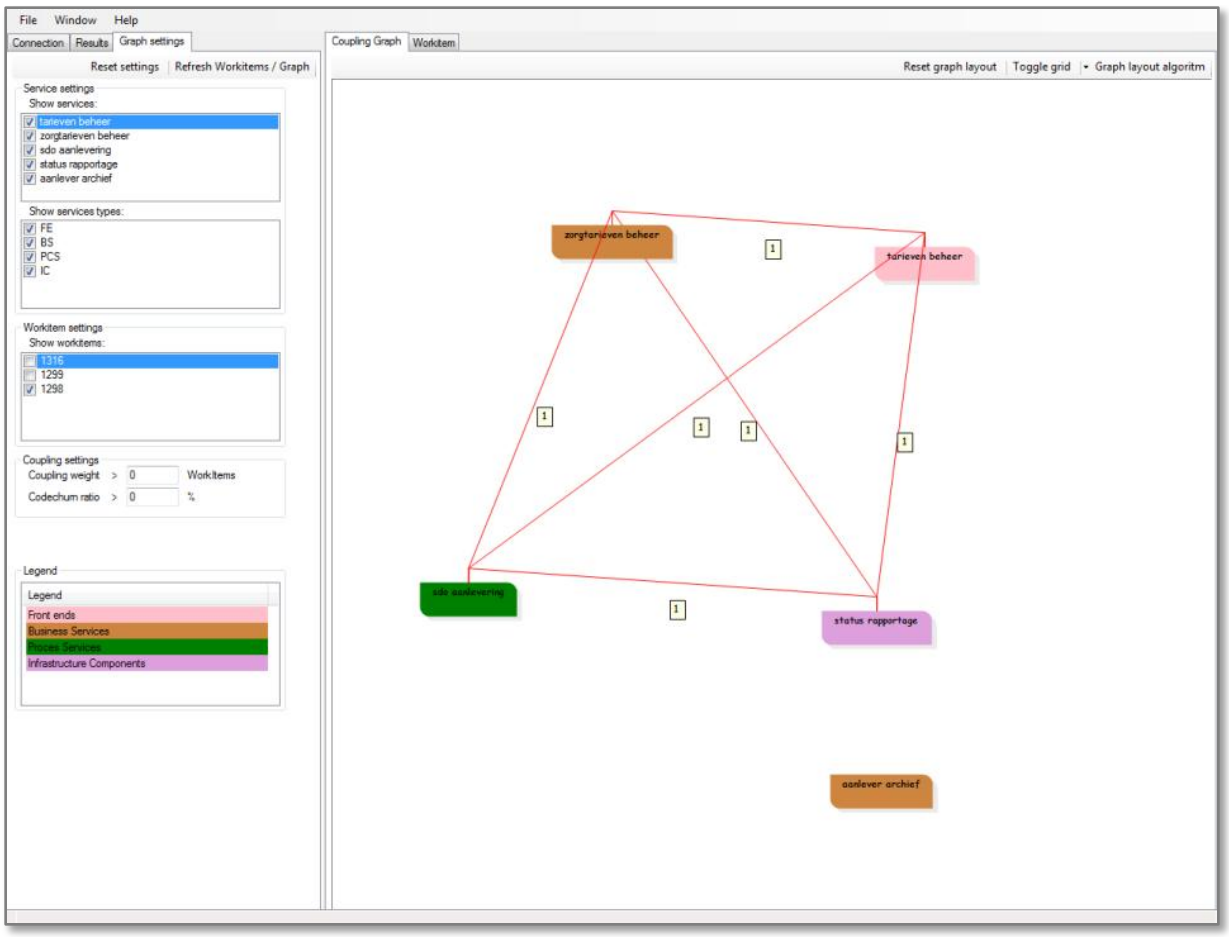
Na het toepassen van de filter bleven de volgende Change Dependencies zichtbaar.

Indicator	Coupling weight	Service x	Service y
CD01	1	FE - tarieven beheer	IC - status rapportage
CD02	1	PCs - sdo aanlevering	IC - status rapportage
CD03	1	BS - zorgtarieven beheer	IC - status rapportage
CD04	1	BS - zorgtarieven beheer	PCs - sdo aanlevering
CD05	1	BS - zorgtarieven beheer	FE - tarieven beheer
CD06	1	PCs - sdo aanlevering	FE - tarieven beheer

Tabel 0-3, Change Dependencies

Graaf

In de graaf zijn de Change Dependencies die voortkomen uit Workitem 1298 zichtbaar.

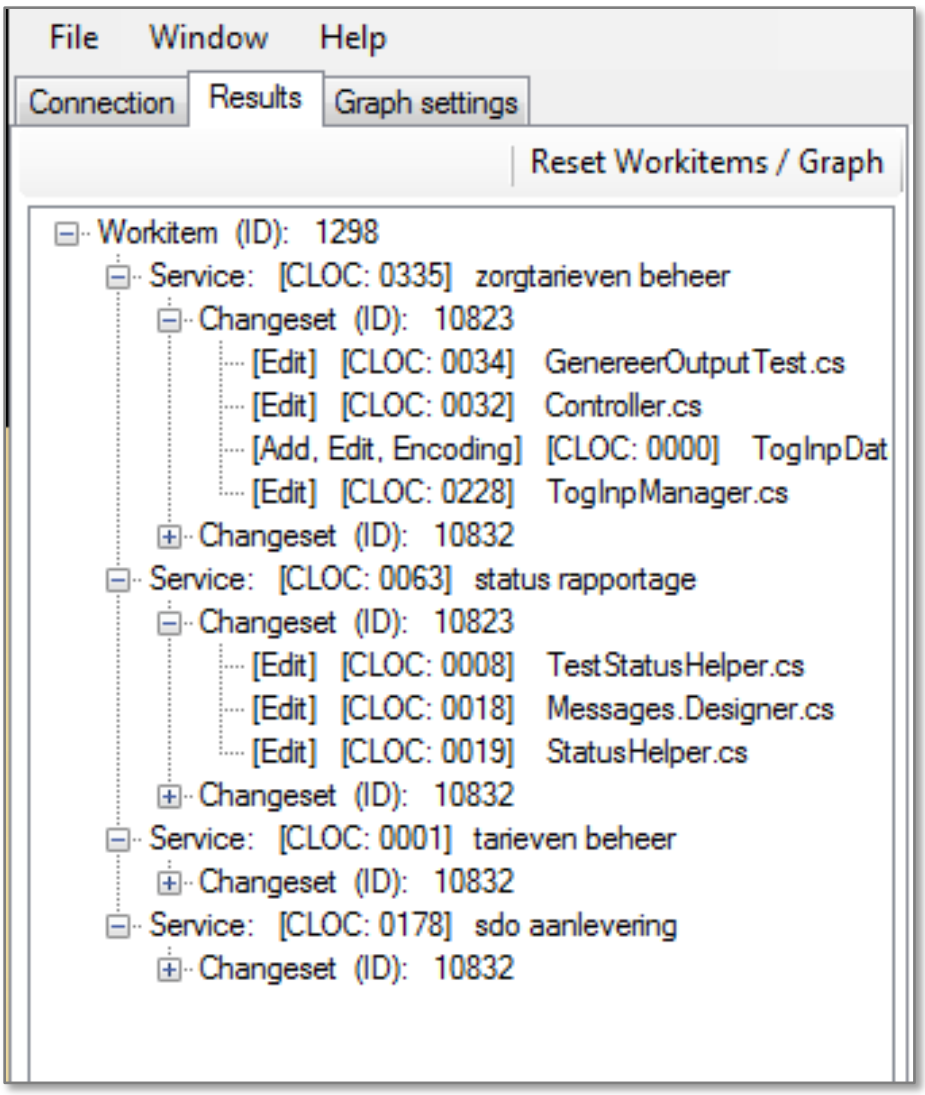


Figuur 0-1, LCA toont Change Dependency graaf

Workitems

Wanneer we kijken naar de Workitems blijken er twee Changesets aan het Workitem 1298 gelinkt te zijn:

- Changeset 10823
- Changeset 10832



Figuur 0-2, Results tab toont Workitems