Onderhoudbaarheid vs. betrouwbaarheid

– een case study –

Pieter Bregman

16 augustus 2012

Universiteit van Amsterdam
Master of Science - Software Engineering

Supervisor:
Jurgen Vinju

Abstract

Onderhoudbaarheid en betrouwbaarheid zijn allebei kwaliteitsaspecten van softwaresystemen volgens ISO/IEC 9126-1. De Software Improvement Group (SIG) heeft een model gemaakt waarmee de onderhoudbaarheid van softwaresystemen berekend kan worden.

In dit onderzoek worden de effecten onderzocht als geprobeerd wordt om met ReSharper van JetBrains de betrouwbaarheid te verbeteren. De effecten van de adviezen van ReSharper worden onderzocht met voorbeelden uit de source code. Hiervan worden de effecten op de betrouwbaarheid en de onderhoudbaarheid onderzocht.

De effecten op de onderhoudbaarheid door het uitvoeren van de adviezen van ReSharper worden onderzocht door de effecten op de onderhoudbaarheidskarakteristieken te meten. Dit zijn het volume van het totale softwaresysteem, de unit grootte in lines of code (LOC) en de cyclomatische complexiteit (CC) van units. Van deze metrieken wordt onderzocht welke invloed zij hebben gehad op de scores die met het model van de SIG zijn berekend.

Uit de onderzoeksresultaten komt naar voren dat de adviezen van ReSharper lang niet altijd het gewenste resultaat met zich mee brachten. De betrouwbaarheid werd niet in alle situaties door de adviezen verbeterd. De onderhoudbaarheidskarakteristieken werden soms ook meer beïnvloed dan nodig was om de beoogde verbetering te realiseren. Deze resultaten lijken aanleiding te geven om de adviezen van ReSharper niet altijd door te voeren.

De onderhoudbaarheid was licht afgenomen na het verhogen van de betrouwbaarheid met de adviezen van ReSharper. Deze afname in de onderhoudbaarheid was zo klein dat het niet geleid heeft tot slechtere scores volgens het model van de SIG.

Voorwoord

In dit document leest u het eindresultaat van de Master Thesis van de Software Engineering opleiding aan de Universiteit van Amsterdam.

Vanuit de opleiding wil ik, in het bijzonder, mijn supervisor Jurgen Vinju bedanken voor de ontvangen begeleiding, de vele interessante discussies, de ontvangen feedback tijdens de uitvoering van het onderzoek en de proces- en kwaliteitsbewaking van mijn scriptie. Hiernaast gaat ook mijn dank uit naar Paul Klint voor de ontvangen feedback op mijn scriptie.

Ook wil ik de mensen bedanken die mijn scriptie hebben willen lezen, van feedback hebben voorzien of op een andere manier een bijdrage aan de totstandkoming van mijn scriptie hebben geleverd.

Naast mijn supervisor zijn dit: Jan & Annet Bregman, Frank & Anneke de Brouwer, Michiel Heerschop & Elly Koeijers en Remigius Barendse.

Heel graag wil ik mijn vriendin Marlies bedanken die mij altijd heeft gesteund en gestimuleerd gedurende de hele opleiding.

Hiernaast wil ik mijn familie en vrienden bedanken voor hun steun die ik van hen heb ontvangen tijdens het uitvoeren van de opleiding.

Mijn werkgever, Van Dorp installaties, wil ik bedanken voor de geboden mogelijkheid om deze opleiding te kunnen volgen.

Ten slotte gaat mijn dank uit naar iedereen die interesse heeft getoond en heeft gevraagd hoe de voortgang van mijn scriptie verliep.

1 Introductie

Het onderhoud aan een softwaresysteem (software maintenance) kan worden samengevat als [1]: “The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.”

De meeste tijd van de ontwikkeling van een softwaresysteem wordt besteed aan het onderhoud daarvan [2][3]. Het onderhoud van een softwaresysteem is daarmee een grote kostenpost en beslaat tussen de 60% en 80% van de totale kosten[4]. Hierdoor is de onderhoudbaarheid van een softwaresysteem een zwaarwegende factor voor het maken van beslissingen over de verdere ontwikkeling. Wanneer de onderhoudbaarheid slecht is kan dit leiden tot het besluit om het softwaresysteem volledig opnieuw te gaan ontwikkelen, in sommige situaties wordt de ontwikkeling zelfs afgebroken.

Dit onderzoek beschrijft of er een relatie bestaat tussen de betrouwbaarheid en de onderhoudbaarheid van een softwaresysteem. Beide zijn volgens de ISO/IEC 9126-1[5] kwaliteitsaspecten van een softwaresysteem. Betrouwbaarheid behelst het correct laten functioneren (de kans op defects [6]) en de herstelcapaciteit van een softwaresysteem. De onderhoudbaarheid van een softwaresysteem beschrijft in hoeverre wijzigingen (het verhelpen van defects, uitbreiden van functionaliteit) aan een softwaresysteem door te voeren zijn.

Als de betrouwbaarheid omhoog gaat, door bijvoorbeeld de kans op defects te verkleinen dan hoeft er minder tijd geïnvesteerd te worden in het op een later tijdstip oplossen van defects. Dit heeft als gevolg dat er minder onderhoud uitgevoerd hoeft te worden, dit zegt niets over de onderhoudbaarheid zelf van een softwaresysteem. Anderzijds als het verhogen van de betrouwbaarheid als gevolg heeft dat de onderhoudbaarheid achteruit gaat, dan is het uitvoeren van onderhoud op het softwaresysteem na de aanpassingen complexer dan ervoor.

Het is dus niet duidelijk welke relatie er bestaat tussen de betrouwbaarheid en de onderhoudbaarheid van een softwaresysteem. Deze onduidelijkheid is de aanleiding voor dit onderzoek, het achterhalen of er een relatie tussen die twee kwaliteitsaspecten bestaat en als die relatie bestaat hoe de kwaliteitsaspecten elkaar beïnvloeden.

Wanneer de aanpassing van het softwaresysteem als doel heeft om de betrouwbaarheid van het softwaresysteem verhogen, dan is er sprake van defensive programming.

> Defensive programming is een vorm van onderhoud waarbij het softwaresysteem wordt aangepast met als doel om de functionele continuïteit ervan te waarborgen.

1.1 Onderzoeksvraag

De hoofdvraag die centraal staat in dit onderzoek is:

> RQ: Bestaat er een relatie tussen onderhoudbaarheid en betrouwbaarheid?

Voor het beantwoorden van deze vraag zijn de volgende deelvragen geformuleerd:

S-RQ1: Zijn onderhoudbaarheid en betrouwbaarheid onafhankelijke kwaliteitsaspecten?
Of verandert de onderhoudbaarheid als de betrouwbaarheid is veranderd? Hoe kan je dit meten?

S-RQ2: Als kwaliteitsaspecten van elkaar afhankelijk zijn, hoe zijn ze dat dan?
Welke invloed wordt er uitgeoefend op de kwaliteitsaspecten als geprobeerd wordt om de betrouwbaarheid te verhogen? Wordt de betrouwbaarheid er echt door verhoogd? Wat is de invloed op de onderhoudbaarheid als geprobeerd wordt om de betrouwbaarheid te verhogen?

S-RQ3: Wat is de impact van een mogelijke afhankelijkheid op de kwaliteitsaspecten?
In welke mate worden de kwaliteitsaspecten van het softwaresysteem beïnvloed als geprobeerd wordt om de betrouwbaarheid te verhogen? Hoeveel invloed heeft dit op de onderhoudbaarheid van het softwaresysteem?

Hoofdstuk 2 beschrijft de onderzoeksmethode. In hoofdstuk 3.1 wordt gerelateerd onderzoek gepresenteerd. Hoofdstuk 3.2 en 3.3 beschrijven hoe de betrouwbaarheid van een softwaresysteem met ReSharper verhoogd kan worden. Hoofdstuk 3.4 en 3.5 beschrijven hoe de onderhoudbaarheid van een softwaresysteem met het SIG model in kaart kan worden gebracht. Hoofdstuk 4 beschrijft de uitvoering van het onderzoek, de resultaten en de analyses van deze resultaten. In hoofdstuk 5 is de samenvattende analyse beschreven en hoofdstuk 6 sluit af met de conclusie.

2 Onderzoeksmethode

In dit onderzoek wordt een analyse uitgevoerd naar de veranderingen die optreden in de onderhoudbaarheid als er wijzigingen worden doorgevoerd waardoor de betrouwbaarheid (het verkleinen van de kans op defects) van een softwaresysteem wordt verhoogd.

Het tegenovergestelde, de veranderingen die zouden kunnen optreden in de betrouwbaarheid als de onderhoudbaarheid wordt verhoogd, vallen buiten de scope van dit onderzoek.

> Voor het onderzoek zullen er codewijzigingen aan 2 softwaresystemen worden doorgevoerd. Deze codewijzigingen hebben als doel om de betrouwbaarheid van het softwaresysteem te verhogen. Na het verhogen van de betrouwbaarheid zullen de effecten op de betrouwbaarheid en de onderhoudbaarheid worden gemeten en geanalyseerd.

Het ene softwaresysteem dat gebruikt wordt is ScrewTurn Wiki[7]. Met ScrewTurn Wiki kunnen wiki's worden aangemaakt en beheerd. ScrewTurn Wiki is een open source .NET 3.5 C# project. Het andere softwaresysteem is EMS. EMS is een .NET 2.0 C# Windows desktop applicatie dat gebruikt wordt door mijn werkgever. Zij gebruiken EMS om energiebeheer voor haar klanten mee uit te voeren. Met EMS kunnen meterstanden geregistreerd worden, verbruiksrapporten worden gemaakt en analyses van het energieverbruik worden uitgevoerd.

Voor het inzichtelijk maken en verhogen van de betrouwbaarheid wordt gebruik gemaakt van ReSharper 5.1 van JetBrains[8]. ReSharper doet voorstellen om de source code van een softwaresysteem aan te passen. Deze voorstellen hebben als doel om de onderhoudbaarheid van het softwaresysteem te verbeteren of om de betrouwbaarheid te verhogen. Tijdens het onderzoek zullen de voorstellen die als doel hebben om de betrouwbaarheid van het softwaresysteem te verhogen, worden toegepast op de softwaresystemen.

Voor het beantwoorden van de deelvragen zullen de effecten op de onderhoudbaarheid – door het verhogen van de betrouwbaarheid met ReSharper – van de softwaresystemen worden onderzocht. Eerst zal onderzocht worden of de onderhoudbaarheid door het uitvoeren van de adviezen van ReSharper is beïnvloed. Hiervoor wordt gebruik gemaakt van metrieken die door het SMM worden gebruikt om onderhoudbaarheid te meten.

Hiernaast zal van de adviezen, die ReSharper geeft om de betrouwbaarheid te verhogen, worden onderzocht op welke manier ze de betrouwbaarheid en de onderhoudbaarheid hebben beïnvloed.

Als laatst zullen de eventuele effecten op de onderhoudbaarheid worden gemeten en geanalyseerd. Dit zal worden gedaan zoals dit staat beschreven in het Software Maintainability Model (SMM)[9] van de Software Improvement Group (SIG)[10].

Uit het SMM zullen metrieken worden gebruikt waarmee de effecten op de onderhoudbaarheid gemeten worden. Deze metrieken zijn de cyclomatische complexiteit (CC) per unit [11], lines of code (LOC) per unit en het totaal aantal lines of code (LOC) van het softwaresysteem.

2.1 Hypothesen:

Voor het beantwoorden van de hoofdvraag is de volgende hypothese opgesteld:

H1: Door het verhogen van de betrouwbaarheid wordt de onderhoudbaarheid slechter.

Bij deze hypothese horen de volgende alternatieve hypotheses:

AH1.1 Door het verhogen van de betrouwbaarheid blijft de onderhoudbaarheid gelijk.

AH1.2 Door het verhogen van de betrouwbaarheid wordt de onderhoudbaarheid beter.

3 Gerelateerd onderzoek en achtergronden

Dit hoofdstuk bevat de literatuurstudie naar ander onderzoek. Hiernaast bevat dit hoofdstuk uitleg over de achtergronden die een rol spelen in dit onderzoek.

Er is op het gebied van onderhoudbaarheid heel veel onderzoek gedaan. Om die reden is er in de literatuurstudie een selectie gemaakt van onderzoeken die gerelateerd zijn aan dit onderzoek.

3.1 Gerelateerd onderzoek

Bart Luijten[12] heeft onderzoek gedaan naar de invloed op de onderhoudbaarheid van softwaresystemen en de afhandeling van issues. Hij toont aan dat er een positieve correlatie bestaat tussen de onderhoudbaarheid berekend volgens het SMM en de afhandelingtijd van issues. Het oplossen van issues in softwaresystemen met hogere onderhoudbaarheid gaat sneller dan in softwaresystemen waar de onderhoudbaarheid lager is.

Dennis Bijlma[13] heeft vervolgens dit onderzoek uitgebreid door meer soorten issues te onderzoeken. Hij heeft ook het aantal indicatoren om de issueafhandeling inzichtelijk mee te maken uitgebreid. Uit zijn resultaten kwam naar voren dat door het toevoegen van de indicatoren en issues, de positieve correlatie tussen de onderhoudbaarheid volgens het SMM en het afhandelen van issues bleef bestaan.

In deze onderzoeken ligt de focus op de positieve correlatie die er lijkt te bestaan tussen de onderhoudbaarheid van softwaresystemen volgens het SMM en het uitvoeren van onderhoudstaken. In dit onderzoek ligt de focus op de invloed die op de onderhoudbaarheid wordt uitgeoefend als geprobeerd wordt om de betrouwbaarheid van softwaresystemen te verbeteren.

In andere onderzoeken [14][15] wordt gebruik gemaakt van Coverity[16] om de betrouwbaarheid van softwaresystemen te analyseren. Coverity is een test platform dat door middel van statische analyses de ontwikkelaar kan wijzen op mogelijke bedreigingen van de betrouwbaarheid van softwaresystemen. Coverity zou een alternatief kunnen zijn voor ReSharper, wat in dit onderzoek is gebruikt om de betrouwbaarheid van softwaresystemen te verhogen.

Ander onderzoek [17] introduceert een model waar de “defect density” van een softwaresysteem vooraf mee voorspeld kan worden. Dit gebeurt door een relatieve code churn metriek te introduceren die afgeleid is van absolute code churn metrieken. Het onderzoek is uitgevoerd op Windows 2003. Daar kwam uit naar voren dat van de onderdelen waarvan voorspeld werd dat er bugs in zouden optreden, en dus de defect density zou toenemen, dit in 89% van de situaties ook bleek te kloppen.

Recent is onderzoek[18] gedaan naar het berekenen van de betrouwbaarheid van softwaresystemen. De onderzoekers presenteren een methode waarmee ze de betrouwbaarheid van softwaresystemen kunnen berekenen. Zij doen dit door met statische code analyse te berekenen hoe groot de kans is dat een softwaresysteem nog werkt na *n* uitvoeringen. Het verwacht aantal uitgevoerde statements, totdat er een fout optreedt in het softwaresysteem, wordt ook berekend.

Deze onderzoeken richten zich op het voorspellen van betrouwbaarheid van softwaresystemen. In het onderzoek over “defect density” wordt dit gedaan door eerder uitgevoerd onderhoud te meten. Aan de hand daarvan kunnen zij defectgevoelige delen binnen de source code aanwijzen. In het andere onderzoek presenteren de onderzoekers een methode voor het voorspellen van de betrouwbaarheid.

Deze onderzoeken richten zich op het berekenen van de betrouwbaarheid en het benoemen van defectgevoelige onderdelen in softwaresystemen. Het verhogen van de betrouwbaarheid, wat in dit onderzoek een belangrijke rol speelt, komt in deze onderzoeken niet aan de orde.

In dit onderzoek staat centraal welke invloed er op de onderhoudbaarheid van softwaresystemen wordt uitgeoefend als geprobeerd wordt om de betrouwbaarheid te verhogen.

3.2 Betrouwbaarheid

Betrouwbaarheid (Reliability) volgens de ISO/IEC 9126-1:

> Once a software system is functioning, as specified, and delivered the reliability characteristic defines the capability of the system to maintain its service provision under defined conditions for defined periods of time

Onderliggende aspecten van betrouwbaarheid:

- Maturity (bedrijfszekerheid)
    - This sub characteristic concerns frequency of failure of the software
- Fault-tolerance (foutbestendigheid)
    - The ability of software to withstand (and recover) from component, or environmental, failure
- Recoverability (herstelbaarheid)
    - Ability to bring back a failed system to full operation, including data and network connections

3.3 Verhogen van de betrouwbaarheid met ReSharper

ReSharper is een plug-in voor de IDE “Visual Studio” van Microsoft. ReSharper kan door het analyseren van de source code adviezen geven waarmee de kwaliteit van het softwaresysteem verhoogd kan worden. ReSharper doet dit door wijzigingen aan de source code voor te stellen die de verbetering van de kwaliteit dienen te realiseren. Dit kunnen zowel wijzigingen zijn om de betrouwbaarheid van het softwaresysteem verhogen, als adviezen die het softwaresysteem beter onderhoudbaar maken.

Tijdens het onderzoek wordt er gekeken naar wijzigingen die ReSharper voorstelt die als doel hebben om de betrouwbaarheid te verhogen, dit zijn de volgende wijzigingen:

- Possible System.NullReferenceException
- Possible System.InvalidCastException
- Possible System.InvalidOperationException

Deze adviezen hebben als doel om de bedrijfszekerheid van het softwaresysteem te vergroten waardoor de betrouwbaarheid wordt verhoogd. De benoemde exceptions kunnen optreden tijdens het uitvoeren van de applicatie maar zullen niet in alle gevallen plaats vinden, dit komt doordat ze afhankelijk zijn van condities.

Omdat deze aanpassingen niet als doel hebben om de werking van het softwaresysteem te veranderen, maar te verbeteren, is er sprake van refactoring.

Refactoring is het proces waarbij een softwaresysteem dusdanig wordt aangepast zodat de uitvoering van het softwaresysteem niet verandert maar er wel verbeteringen aan de source code van het softwaresysteem worden gerealiseerd[19].

3.3.1 Possible System.NullReferenceException

Een possible System.NullReferenceException kan optreden als een softwaresysteem aannames doet over de gegevens die het verwerkt. Deze gegevens kunnen bijvoorbeeld een document zijn, user input of een memory stream.

Bijvoorbeeld het inlezen van een Node uit een XML bron, wanneer de Node op basis van een attribuut of positie wordt ingelezen dan kan het in bepaalde gevallen voorkomen dat de Node null is. Dit kan komen omdat de Node niet meer op die positie staat of een ander attribuut heeft gekregen.

Wanneer het softwaresysteem hierna de gegevens aanspreekt, dan blijken deze null te zijn en treedt een NullReferenceException op. ReSharper adviseert in de meeste gevallen om het mogelijk optreden van deze exception te ondervangen door vooraf te controleren of de variabele die de gegevens zou moeten bevatten niet null is. In sommige gevallen kan ReSharper de exception vervangen door een casting uit te voeren. Volgens ReSharper kan het controleren of een variabele niet null is bereikt worden door de variabele met een IF statement hierop te controleren.

```
if (x != null) {
    x.someMethod(); // <-- NullReferenceException had opgetreden als x wel null was geweest
}
```

Dit statement kan eventueel worden uitgebreid met een ELSE block, hierin kan eventuele foutafhandeling plaatsvinden, een custom exception worden gegenereerd of een melding worden teruggestuurd naar de “caller”. Als de ontwikkelaar een ELSE block implementeert, dan kan het softwaresysteem de ontstane situatie ondervangen. Hierdoor kan ook de herstelbaarheid van het softwaresysteem worden verbeterd.

ReSharper kan niet in alle gevallen een oplossing voorstellen om een mogelijke NullReferenceException te voorkomen. De ontwikkelaar zal dan zelf de source code moeten aanpassen om de mogelijke NullReferenceException te voorkomen.

3.3.2 Possible System.InvalidCastException

Een possible System.InvalidCastException treedt op als een variabele expliciet gecast wordt naar een type dat niet ondersteund wordt door het oorspronkelijke type van de variabele.

Problemen met casts treden meestal op bij het gebruik van ArrayLists, als daar waardes aan toegevoegd worden die op een bepaald type lijken maar het niet zijn, kunnen er problemen ontstaan tijdens het expliciet casten hiervan.

Voorbeeld:

Als aan een ArrayList de waarde "100" (inclusief quotes) wordt toegevoegd. Als de waarde wordt gecast naar een int dan resulteert dit in een InvalidCastException omdat de waarde oorspronkelijk van het type string is.

Deze exceptions komen sinds de introductie van "lists" en "dictionaries" veel minder voor, dit komt omdat daar geen casting meer hoeft te worden gedaan om met de waardes te kunnen werken.

Soms treedt er een probleem met casting op omdat de ontwikkelaar een verkeerde aanname heeft gedaan over het type van de variabele. Als deze casting zich achter uitzonderlijke voorwaarden bevindt is de kans dat deze exception zich snel voordoet niet groot.

3.3.3 Possible System.InvalidOperationException

Een possible System.InvalidOperationException kan optreden als een softwaresysteem bezig is met het doorlopen van gegevens terwijl die gegevens op dat moment worden gewijzigd. Een voorbeeld[1] van een InvalidOperationException is hieronder weergegeven:

```
foreach (TreeNode treeNode in treeView1.Nodes) {
        if (treeNode.Nodes.Count == 0) {
                treeNode.Remove(); // ← Veroorzaakt een InvalidOperationException
        }
}
```

Voorbeeld:

Tijdens het doorlopen (met een foreach()) van een verzameling van TreeNodes in een TreeView is het de bedoeling dat TreeNodes met eigenschap X worden verwijderd.

Nadat de eerste TreeNode wordt verwijderd is de verzameling niet meer hetzelfde en treedt de InvalidOperationException op omdat de verzameling zich in alleen-lezen modus bevindt tijdens het doorlopen.

Doordat het verwijderen van de objecten in de verzameling afhankelijk is van specifieke voorwaarden, kan op voorhand niet altijd worden bepaald of en wanneer de exception zal optreden. Dit wordt nog moeilijker als de uitkomst van de voorwaarden bepaald wordt door gegevens uit andere softwaresystemen.

Het optreden van bovenstaande InvalidOperationException kan worden voorkomen door de referenties van de TreeNodes die verwijderd moeten worden te verzamelen in een aparte lijst. Na het afronden van de foreach procedure, kan de verzameling weer gemodificeerd worden. Hierna kan de aparte lijst worden doorlopen om de TreeNodes alsnog te verwijderen.

3.4 Onderhoudbaarheid

Onderhoudbaarheid (Maintainability) volgens de ISO/IEC 9126-1:

> The ability to identify and fix a fault within a software component is what the maintainability characteristic addresses

[1] http://www.dotnetperls.com/invalidoperationexception

Onderliggende aspecten van onderhoudbaarheid:

- Analyzability (analyseerbaarheid)
  - Characterizes the ability to identify the root cause of a failure within the software.
- Changeability (wijzigbaarheid)
  - Characterizes the amount of effort to change a system.
- Stability (stabiliteit)
  - Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.
- Testability (testbaarheid)
  - Characterizes the effort needed to verify (test) a system change.

3.5 Onderhoudbaarheid meten met het Software Maintainability Model (SMM)

De SIG (Software Improvement Group) heeft een model[9] gemaakt waarmee de onderhoudbaarheid van een softwaresysteem berekend kan worden. Het model voorspelt aan de hand van verzamelde metrieken, die gebaseerd zijn op karakteristieken van de source code, hoe onderhoudbaar een softwaresysteem is. De verzamelde metrieken worden door het model geanalyseerd om vervolgens de onderhoudbaarheid van het softwaresysteem uit te drukken in een score. De mogelijke scores zijn (van hoog naar laag): “++”, “+”, ”o”, “-” en “--”.

De meetbare karakteristieken zijn volledig gebaseerd op de source code van het softwaresysteem. De metrieken die verzameld worden zijn: het totale volume van het softwaresysteem, de complexiteit van een unit[2], het aantal blokken met duplicaten/clones, de grootte van een unit en de aanwezigheid van tests voor een unit.

Van iedere karakteristiek heeft de SIG bepaald op welke eigenschappen van de onderhoudbaarheid ze van invloed zijn. Het verhogen van de betrouwbaarheid door het uitvoeren van de adviezen van ReSharper kunnen invloed hebben op de onderhoudbaarheidskarakteristieken van de source code. Dit zijn dezelfde karakteristieken die door het SMM worden gebruikt om de onderhoudbaarheid te berekenen.

De karakteristieken “Volume”, “Unit size”, en “Complexity per unit” uit het SMM zijn tijdens dit onderzoek gebruikt om de invloed op de onderhoudbaarheid te meten. De karakteristieken “Duplication” en “Unit testing” zijn niet gebruikt in dit onderzoek, dit is gedaan omdat het verhogen van de betrouwbaarheid geen invloed uitoefent op deze karakteristieken.

[2] Een unit is in .NET een methode binnen een class

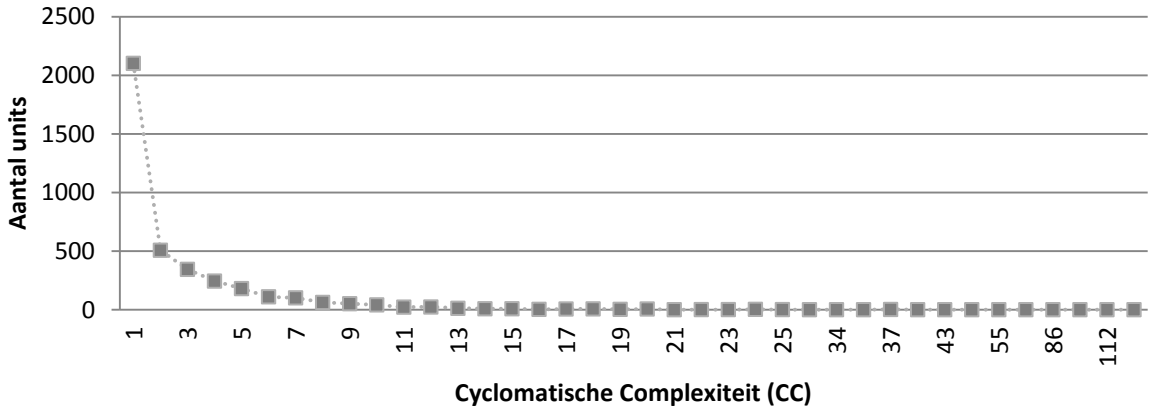
4 Onderhoudbaarheid vs. betrouwbaarheid

Voor het beantwoorden van de deelvragen zullen de softwaresystemen worden aangepast om te proberen de betrouwbaarheid hiervan te verhogen. De effecten op de onderhoudbaarheid van de softwaresystemen zullen vervolgens worden gemeten en geanalyseerd. In Tabel 1 staan de karakteristieken van de gebruikte softwaresystemen opgenoemd.

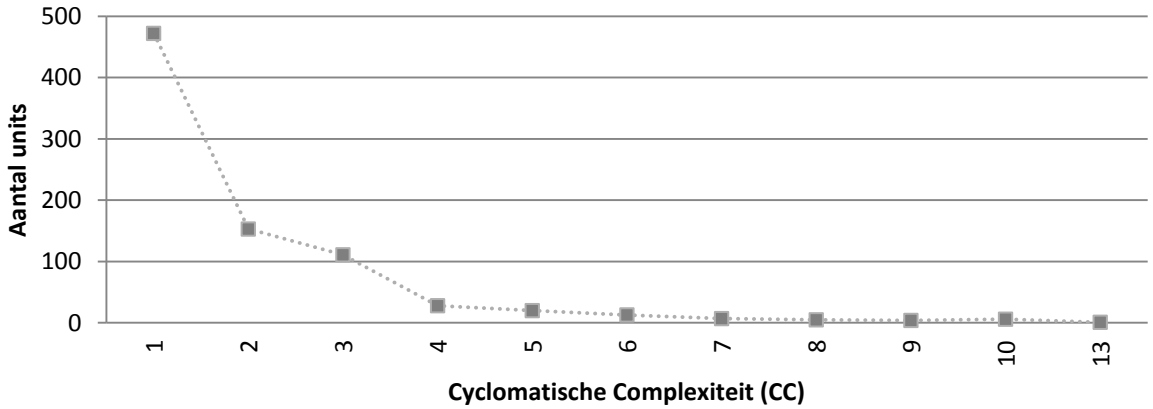
Softwaresysteem	Units	Totaal LOC	LOC van de grootste unit	CC van meest complexe unit
ScrewTurn Wiki	3885	32899	551	183
EMS	820	3567	46	13

Tabel 1 – Karakteristieken van de softwaresystemen

In Figuur 1 en Figuur 2 is te zien hoe de CC zich verhoudt ten opzichte van het aantal units in het softwaresysteem. De punten in de grafiek geven het aantal units per CC weer.



Figuur 1 – ScrewTurn Wiki: Units per Cyclomatische Complexiteit (CC) – voor aanpassingen

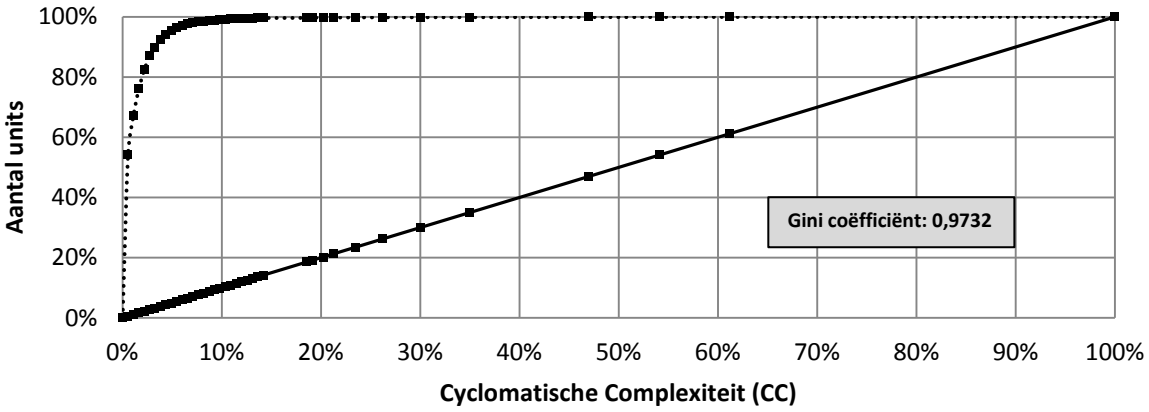


Figuur 2 – EMS: Units per Cyclomatische Complexiteit (CC) – voor aanpassingen

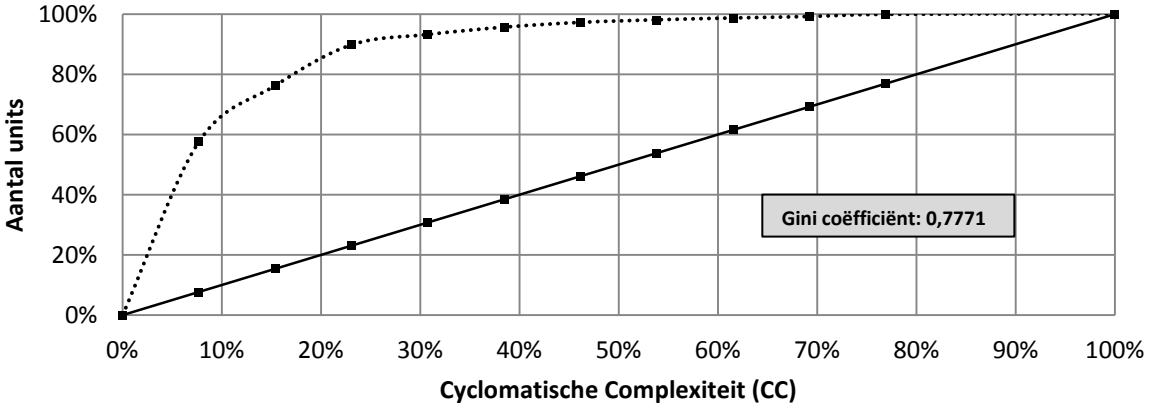
Om na de aanpassingen te kunnen meten of de onderhoudbaarheid is veranderd is er een Lorenz Curve met de Gini coëfficiënt (Figuur 3 en Figuur 4) van de softwaresystemen toegevoegd. De Gini coëfficiënt wordt meestal gebruikt om de gelijkheid van een verdeling mee uit te drukken.

De Lorenz Curve met de Gini coëfficiënt zullen in dit onderzoek gebruikt worden om te kijken in hoeverre deze voor en na de aanpassingen van elkaar verschillen. Hiermee kan worden vastgesteld of er een grote verandering in de verdeling van de CC heeft plaats gevonden en kan de invloed op de onderhoud door het doorvoeren van de wijzigingen worden gemeten.

De Gini coëfficiënt geeft de gelijkheid van een verdeling weer.
De Gini coëfficiënt wordt uitgedrukt in een getal tussen de 0 en 1.
De waarde 0 staat voor een perfecte gelijkheid en 1 voor een perfecte ongelijkheid



Figuur 3 – ScrewTurn Wiki: Lorenz Curve en Gini coëfficiënt – voor aanpassingen



Figuur 4 – EMS: Lorenz Curve en Gini coëfficiënt – voor aanpassingen

4.1 Zijn onderhoudbaarheid en betrouwbaarheid onafhankelijke kwaliteitsaspecten?

Zijn onderhoudbaarheid en betrouwbaarheid onafhankelijke kwaliteitsaspecten? Verandert de onderhoudbaarheid als de betrouwbaarheid is veranderd? Hoe kan je dit meten? Voor het beantwoorden van deze vraag is het van belang om de effecten op de onderhoudbaarheid door het verhogen van de betrouwbaarheid in kaart te brengen.

4.1.1 Methode

Van beide softwaresystemen zal met ReSharper worden geanalyseerd welke mogelijke exceptions kunnen optreden. Van iedere exception zal op het advies van ReSharper worden geprobeerd om de source code aan te passen zodat de mogelijke exception niet meer kan optreden. Als ReSharper geen advies kan geven, zal de source handmatig code worden aangepast, zodat de mogelijke exception niet meer kan optreden.

De mogelijke exceptions waar ReSharper geen oplossing kan bieden, zijn apart gemeten. Na het oplossen van de mogelijke exceptions zal gekeken worden of de onderhoudbaarheid hierdoor is beïnvloed. Dit wordt gedaan door de volgende metrieken te onderzoeken:

- Zijn er units waarvan de CC is gestegen?
- Zijn er units waarvan het aantal LOC is toegenomen?
- Is het totaal aan LOC van het softwaresysteem toegenomen?

4.1.2 Resultaten

De analyse met ReSharper toonde aan dat er in beide softwaresystemen mogelijke exceptions konden optreden. Tabel 2 geeft per type exception weer hoeveel er volgens ReSharper konden optreden in de softwaresystemen.

Softwaresysteem	NullReferenceExceptions	InvalidCastExceptions	InvalidOperationExceptions
ScrewTurn Wiki	82	0	0
EMS	30	1	0

Tabel 2 – Analyse van de softwaresystemen met ReSharper

ReSharper heeft in ScrewTurn Wiki over 59 van de mogelijke NullReferenceExceptions een advies kunnen geven waardoor de exception niet meer kon optreden. Toen deze mogelijke NullReferenceExceptions waren opgelost kon ScrewTurn Wiki niet meer worden gecompileerd. Dit kwam doordat bepaalde aanpassingen er voor hadden gezorgd dat sommige units geen return value meer hadden.

Om ScrewTurn Wiki weer compileerbaar te maken zijn de units aangepast door een null value te retourneren, hierdoor kon het softwaresysteem weer compileerbaar worden gemaakt. Een voorbeeld van een dergelijke aanpassing is te zien in Figuur 5.

Voor:

```
return object.method();
```

Na:

```
if (object != null) { // ← Door ReSharper voorgesteld
   return object.method();
}
return null;
```

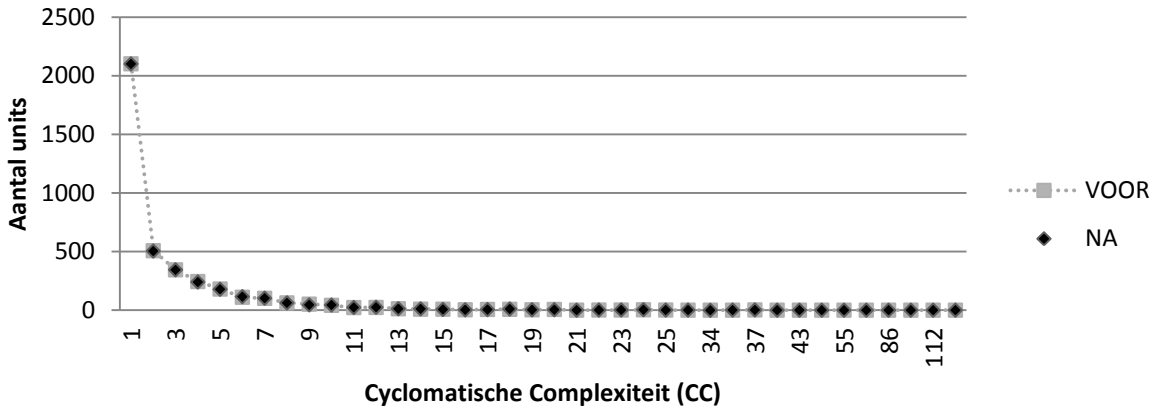
Figuur 5 – Aanpassen van unit om deze weer compileerbaar te maken

De overgebleven 23 mogelijke NullReferenceExceptions in ScrewTurn Wiki zijn handmatig opgelost. In de meeste van deze gevallen kon niet voorkomen worden dat de NullReferenceException niet

meer kon optreden door het advies van ReSharper. In sommige gevallen kon ReSharper geen oplossing aandragen omdat de werking van het softwaresysteem dan niet meer gegarandeerd kon worden. Voorbeeld van een mogelijke NullReferenceException die niet opgelost kon worden met ReSharper (aangetroffen in ScrewTurn Wiki):

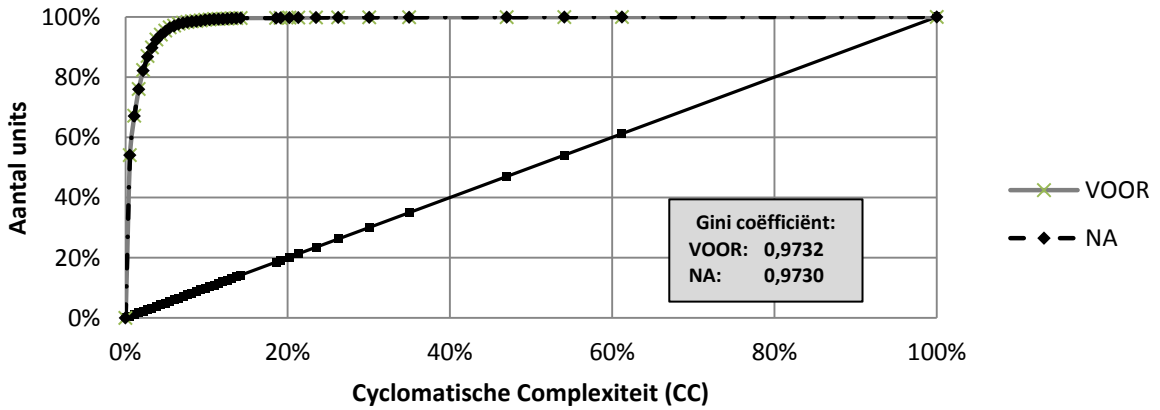
```
if (result != null) {
   // Does something with result
} else {
   // Also does something with result
}
```

Figuur 6 toont de verhouding van het aantal LOC ten opzichte van de CC van ScrewTurn Wiki nadat alle mogelijke exceptions opgelost waren. De verhouding voor de aanpassingen is ook weergegeven in Figuur 6, in de grafiek is nauwelijks verschil waar te nemen.



Figuur 6 – ScrewTurn Wiki: Units per Cyclomatische Complexiteit (CC) – na aanpassingen

In totaal zijn 54 units in CC toegenomen. Het totaal LOC van het softwaresysteem is gestegen naar 33043, dit is een toename van 144 LOC.



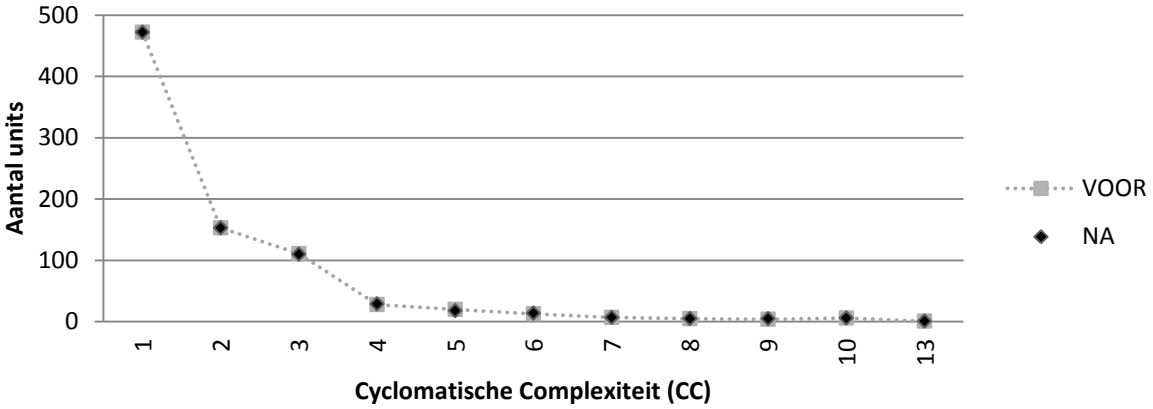
Figuur 7 – ScrewTurn Wiki: Lorenz Curve en Gini coëfficiënt – na aanpassingen

De Lorenz Curve en Gini coëfficiënt van de ScrewTurn Wiki zien er na de aanpassingen uit zoals weergegeven in Figuur 7. De verhouding voor de aanpassingen is ook weergegeven in Figuur 7, in deze grafiek is ook nauwelijks verschil waar te nemen.

Bij het afhandelen van de mogelijke exceptions in EMS, bleek dat 23 van de 30 mogelijke NullReferenceExceptions, zich in gegenereerde source code bevonden. Deze source code wordt door een generator van buiten het softwaresysteem gegenereerd, deze source code wordt door EMS gebruikt om de tabellen van de database waarmee EMS communiceert te “mappen”.

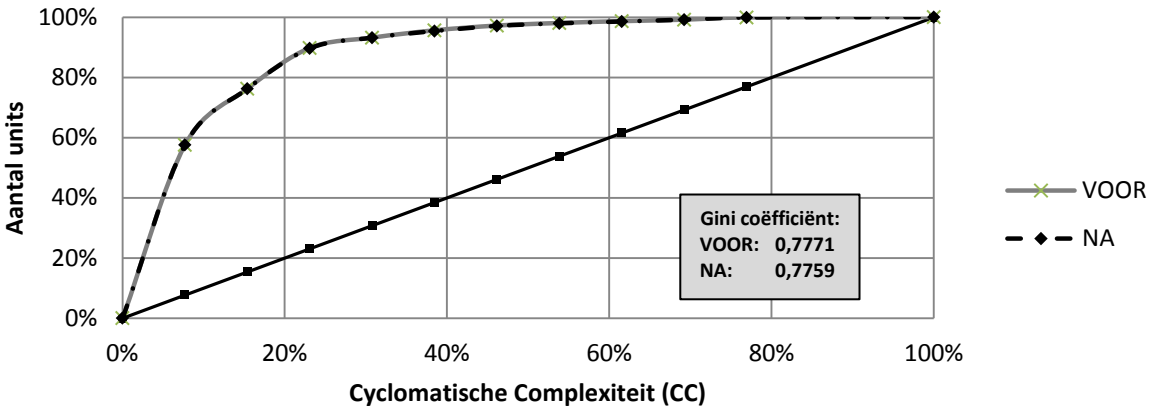
Van de 7 mogelijke NullReferenceExceptions die overbleven konden er 2 met ReSharper opgelost worden door een typecasting van een variabele te doen. De overige 5 mogelijke NullReferenceExceptions konden worden opgelost door het plaatsen van een IF statement.

Tijdens het aanpassen van EMS konden alle mogelijke NullReferenceExceptions in de eerste scan worden opgelost. De InvalidCastException moest handmatig worden opgelost, omdat ReSharper hier geen oplossing voor kon aandragen. Na het oplossen van alle mogelijke exceptions bedroeg het totaal aantal LOC van EMS 3574, dit is een toename van 7 LOC.



Figuur 8 – EMS: Units per Cyclomatische Complexiteit (CC) – na aanpassingen

De effecten op de LOC ten opzichte van de CC staan in Figuur 8 weergegeven. Figuur 9 toont de Lorenz Curve en Gini coëfficiënt van EMS na de aanpassingen. Net zoals bij ScrewTurn Wiki is in de grafieken nauwelijks verschil waar te nemen tussen voor en na de aanpassingen.



Figuur 9 – EMS: Lorenz Curve en Gini coëfficiënt – na aanpassingen

Door het oplossen van de mogelijke NullReferenceExceptions zijn alle aangepaste units in CC toegenomen en is er geen unit in CC afgenomen. Bij een paar units in EMS zijn units in CC gelijk

gebleven omdat de mogelijke NullReferenceExceptions konden worden voorkomen met een typecasting. In Tabel 3 is deze groei in CC weergegeven.

Toename in Cyclomatische Complexiteit (CC)	1	2	3	4	5	6	7	8	9	10
Aantal units in ScrewTurn Wiki	29	17	4	2	1	0	0	0	0	1
Aantal units in EMS	6	0	0	0	0	0	0	0	0	0

Tabel 3 – Aantal units per CC dat toenam na het verhelpen van mogelijke exceptions

Per CC niveau is aangegeven hoeveel units met dit niveau zijn toegenomen.

4.1.3 Analyse

Door het voorkomen van de mogelijke exceptions zijn er in zowel ScrewTurn Wiki als in EMS units in CC gestegen. Het oplossen van de mogelijke exceptions heeft er ook voor gezorgd dat het totaal LOC van de softwaresystemen is toegenomen.

Het verhogen van de betrouwbaarheid van de softwaresystemen heeft invloed uitgeoefend op de onderhoudbaarheid van de softwaresystemen. Het wegnemen van de mogelijke exceptions heeft units in CC doen stijgen, maar niet doen niet afnemen. Er zijn units toegenomen in LOC en daarmee is ook de LOC van de gehele softwaresystemen toegenomen.

Tijdens het oplossen van een mogelijke exception verandert in de meeste gevallen de CC van de unit. Hierdoor neemt het aandeel units/LOC van een bepaalde CC af en neemt een hierop volgend aandeel van units/LOC van een bepaalde CC toe. Het aandeel LOC van de CC neemt met het aantal LOC van de unit af en wordt toegevoegd (plus het aantal LOC om de mogelijke exception te voorkomen) aan het CC aandeel waar de unit naar toe veranderd is.

Voorbeeld:
Er zijn P units met CC: X en er zijn Q units met CC: Y.
Een unit met CC: X wijzigt in CC: Y.
Dan zijn er nu P – 1 units met CC: X en Q + 1 units met CC: Y.

4.1.4 Discussie: is de betrouwbaarheid door de wijzigingen echt verhoogd?

Met ReSharper zijn de plaatsen binnen de softwaresystemen, waar exceptions zich konden voordoen, in kaart gebracht. Bij de exceptions heeft ReSharper geprobeerd zelf een oplossing aan te dragen om de kans van het optreden van de exception te voorkomen. Bij de NullReferenceExceptions adviseert ReSharper in bijna alle gevallen om de desbetreffende variabele eerst te controleren of deze niet null is. Deze controle wordt gedaan door een IF statement toe te voegen.

Hierdoor is het softwaresysteem niet automatisch betrouwbaarder geworden. De mogelijkheid van het optreden van de NullReferenceException is weg genomen, maar het softwaresysteem onderneemt verder geen actie. Om de betrouwbaarheid echt te verhogen zou het softwaresysteem ook een implementatie moeten krijgen die wordt uitgevoerd als geconstateerd wordt dat de mogelijke exception is voorkomen.

4.2 Als kwaliteitsaspecten van elkaar afhankelijk zijn, hoe zijn ze dat dan?

Welke invloed wordt er uitgeoefend op de kwaliteitsaspecten als met ReSharper wordt getracht om de betrouwbaarheid te verhogen? Wordt de betrouwbaarheid echt verhoogd, of treedt er ook vermindering van de betrouwbaarheid op? Op welke manier wordt de onderhoudbaarheid beïnvloed, door de adviezen van ReSharper, in zowel positieve als negatieve zin?

Voor het beantwoorden van deze deelvraag is het van belang om inzicht te krijgen in de effecten op de kwaliteitsaspecten die de voorstellen van ReSharper hebben. Welke uitwerking hebben de adviezen op de daadwerkelijke betrouwbaarheid van de softwaresystemen, wordt de betrouwbaarheid er echt door verhoogd? Welke invloed hebben ze op de onderhoudbaarheid?

Met andere woorden, wat verandert er aan de betrouwbaarheid en de onderhoudbaarheid door de voorstellen van ReSharper?

4.2.1 Methode

De units waar ReSharper mogelijke exceptions heeft gevonden, worden onderzocht. Hierbij zal gekeken worden welke voorstellen ReSharper heeft gedaan om het optreden van de mogelijke exception te voorkomen.

Van deze voorstellen zal bekeken worden of ze de betrouwbaarheid van het softwaresysteem daadwerkelijk hebben verbeterd en hoe die verbetering is bereikt. Hiernaast zal onderzocht worden of de wijzigingen ook negatieve effecten hebben gehad op de betrouwbaarheid van het softwaresysteem. Vervolgens zal dit ook worden gedaan voor de effecten op de onderhoudbaarheid van het softwaresysteem.

In de resultaten zijn een aantal source code fragmenten, afkomstig uit de softwaresystemen, weergeven. In deze fragmenten is met behulp van ReSharper het optreden van een mogelijke exception voorkomen. Van elk source code fragment zal worden onderzocht welke effecten ze hebben gehad op de betrouwbaarheid en de onderhoudbaarheid.

4.2.2 Resultaten

Per source code fragment worden de resultaten apart gepresenteerd. De toegevoegde of gewijzigde source code is **grijs gearceerd** in de source code fragmenten.

4.2.2.1 Source code fragment 1

In dit source code fragment (Tabel 4) is door ReSharper een mogelijke NullReferenceException gevonden. ReSharper heeft om het optreden van deze exception te voorkomen, geadviseerd om voordat “Success” van “match” wordt aangesproken eerst te controleren of “match” niet null is.

```
1  private void endPageDownload(IAsyncResult ar) {
2     for(int i = 0; i < pageList.Items.Count; i++) {
3        if(pageList.Items[i].Selected) {
4           string url = "";
5           Regex textarea = null;
6           Match match = null;
7           ITranslator translator = null;
8           if(lstWiki.SelectedValue.ToUpperInvariant() == "MEDIA") {
9              url = txtWikiUrl.Text + "?title=" + pageList.Items[i].Value + "&action=edit";
10             textarea = new
               Regex(@"(?<=(\<textarea([^>])*?)\>)(.|\s)+?(?=(\<\/textarea\>))");
11             translator = new Translator();
12          }
13          if(lstWiki.SelectedValue.ToUpperInvariant() == "FLEX") {
14             if(txtWikiUrl.Text.EndsWith("/")) url = txtWikiUrl.Text +
               "wikiedit.aspx?topic=" + pageList.Items[i].Value;
15             else url = txtWikiUrl.Text + "/wikiedit.aspx?topic=" +
               pageList.Items[i].Value;
16             textarea = new Regex(@"(?<=(\<textarea
               class=\'EditBox\'([^>])*?)\>)(.|\s)+?(?=(\<\/textarea\>))");
17             translator = new TranslatorFlex();
18          }
19          try {
20             if (textarea != null) {
21                match = textarea.Match(this.PageRequest(url));
22             }
23             if (match != null) {
24                if(match.Success) {
25                   if (translator != null) {
26                      string text = translator.Translate(match.Value.Replace("&lt;",
                        "<").Replace("&gt;", ">").Replace("&quot;", @""""));
27                      string pageName = this.pageList.Items[i].Value.Replace(":",
                        "_").Replace("/", "_").Replace(@"\", "_").Replace('?', '_');
28                      string pageTitle = this.pageList.Items[i].Value;
29                      Log.LogEntry("Page " + pageName + " created with import whole wiki",
                        EntryType.General, "import");
30                      PageInfo pg = Pages.FindPage(pageName);
31                      SaveMode saveMode = SaveMode.Backup;
32                      if(pg == null) {
33                         Pages.CreatePage(null as string, pageName);
34                         pg = Pages.FindPage(pageName);
35                         saveMode = SaveMode.Normal;
36                      }
37                      Log.LogEntry("Page create requested for " + pageName,
                        EntryType.General, "import");
38                      Pages.ModifyPage(pg, pageTitle, "import", DateTime.Now, "", text,
                        null, null, saveMode);
39                   }
40                   this.pageList.Items.Remove(this.pageList.Items[i]);
41                }
42             }
43          }
44          catch(WebException) { }
45       }
46    }
47    pageList.Visible = true;
48    lblPageList.Text = "Import completed!";
49 }
```

Tabel 4 – ScrewTurn Wiki: ScrewTurn.Wiki.Import.Import.endPageDownload

Verhoging van de betrouwbaarheid

"match" en "textarea" worden beiden met null geïnitieerd. Of "match" null blijft is afhankelijk van of "textarea" null blijft. Dit is weer afhankelijk van de waarde van "selectedValue" in "lstWiki". Door eerst te controleren of "match" niet null is kan de NullReferenceException op "match" niet meer optreden.

Vermindering van de betrouwbaarheid

Door de aanpassing wordt gecontroleerd of "match" niet null is, voordat er met "match" wordt gecommuniceerd. Er is vanuit ReSharper geen oplossing voorgesteld voor een afhandeling als

“match” wel null blijkt te zijn. Hierdoor vindt er geen terugkoppeling plaats naar de “caller” en er wordt ook geen actie ondernomen binnen de unit zelf.

Verhoging van de onderhoudbaarheid

De originele source code gaat er vanuit dat de eigenschap “Success” van “match” aangesproken kan worden. Het toevoegen van de controle of “match” niet null is, zorgt ervoor dat er geen verwarring meer is of het aanspreken van de “Success” eigenschap wel mogelijk is.

Vermindering van de onderhoudbaarheid

Het niet null zijn van “match” is afhankelijk van of “textarea” niet null is, deze controle vindt plaats in regel 20, direct boven de aanpassing die geadviseerd is door ReSharper. Als de controle op “Success” en de implementatie die daarop volgt in de implementatie na de controle op “textarea” vanaf regel 21 zou worden geplaatst, dan is kans dat “match” null kan zijn uitgesloten. Hierdoor verandert de uitvoering van de sofware niet ten opzichte van het voorstel van ReSharper, maar worden de CC en het aantal LOC van de unit niet verhoogd.

Het toevoegen van het IF statement heeft ervoor gezorgd dat er 3 genest IF statements onder elkaar staan. Hierdoor is het oplossen van de mogelijke NullReferenceException ten koste gegaan van de analyseerbaarheid van de source code. Dit effect had voorkomen kunnen worden door eerst de controles tegenovergesteld uit te voeren voorzien van een “continue” statement. Deze techniek wordt door Fowler aangeduid als de “Replace Nested Conditional with Guard Clauses”[19].

Eindafweging

Het is mogelijk dat de ontwikkelaar de code heeft geschreven met de aanname dat de waarde van “selectedValue” van “lstWiki” altijd een van de geteste waardes in de condities is. Hierdoor is de kans dat textarea null blijft uitgesloten en zou de NullReferenceException nooit kunnen optreden. Deze conclusie is alleen niet uit de source code te herleiden en hoeft daarom ook niet waar te zijn.

Hiernaast is het mogelijk dat er in de toekomst een extra waarde toegekend wordt aan “selectedValue” van “lstWiki”. Dit zou ertoe kunnen leiden dat het vanaf dan moment niet meer zo hoeft te zijn dat de een van de geteste condities altijd waar is.

Door het toevoegen van het IF statement waarmee wordt gecontroleerd of “match” niet null is, wordt de NullReferenceException voorkomen en is waarneembaar dat er rekening mee is gehouden dat “match” null zou kunnen blijven. Als het niet de verwachting is dat “match” null kan blijven, dan is het implementeren van een ELSE statement waarin de afhandeling van een ontstane NullReferenceException wordt gedaan, een aanbeveling.

4.2.2.2 Source code fragment 2

In dit source code fragment (Tabel 5) wordt in EMS de waarden van een CSV bestand als “samples” toegevoegd aan de database Repository van EMS. ReSharper heeft op regel 17 een mogelijke NullReferenceException gevonden in deze unit. Volgens ReSharper kan het voorkomen dat “line” op die regel in sommige gevallen null is. Om te voorkomen dat de NullReferenceException kan optreden adviseert ReSharper om, voordat met “line” wordt gecommuniceerd, te controleren of “line” niet null is.

```
1  void ImportCSV(string csvPath)
2  {
3     HaveSamplelist();
4
5     CultureInfo cultureInfo = CultureInfo.GetCultureInfo("nl-NL");
6
7     using (StreamReader sr = File.OpenText(csvPath))
8     {
9        List<Sample> newSamples = new List<Sample>();
10
11       int linenum = 0;
12       while (!sr.EndOfStream) {
13          string line = sr.ReadLine();
14          linenum++;
15          if (line != null) {
16
17             string[] values = line.Split(new char[]{';'},
                StringSplitOptions.RemoveEmptyEntries);
18
19             if (values.Length != 2) {
20                MessageBox.Show("Er worden 2 kolommen in het komma gescheiden bestand
                  verwacht (tijd, waarde)");
21                foreach (Sample s in newSamples) s.Delete();
22                return;
23             }
24
25             Sample sam = Repository.DataSet.SampleTable.New();
26             try {
27                sam.Sampletime = DateTime.Parse(values[0]);
28                sam.Value = double.Parse(values[1], cultureInfo);
29                newSamples.Add(sam);
30             }
31             catch (Exception)
32             {
33                MessageBox.Show("Fout bij lezen van CSV op lijn " + linenum.ToString() +
                  "\n" + line);
34                foreach (Sample s in newSamples) s.Delete();
35                return;
36             }
37          }
38       }
39
40
41       // Speedup, this causes all samples to be committed in a single transaction
42       Repository.DataSet.Connection.BeginConnection();
43       samplesChildListBinding.IgnoreUpdate=true;
44       foreach (Sample s in newSamples) {
45          s.Samplelist = component.InputSamplelist;
46          s.Commit();
47       }
48       Repository.DataSet.Connection.EndConnection();
49       samplesChildListBinding.IgnoreUpdate=false;
50       samplesChildListBinding.Update();
51    }
   }
```

Tabel 5 – EMS: EmsClient.DataEdit.ComponentSamplelistTableDlg.ImportCSV – na aanpassingen

Verhoging van de betrouwbaarheid

In dit voorbeeld is er geen sprake van verhoging van de betrouwbaarheid. Met het IF statement is wel voorkomen dat “line” niet gelijk aan null kan zijn als gevolg van “sr.ReadLine()”. De mogelijke NullReferenceException die hiermee wordt voorkomen, zou zijn opgetreden als “sr.ReadLine()” wordt aangeroepen terwijl het einde van de stream “sr” al is bereikt. Maar of het einde van stream “sr” is bereikt wordt al in regel 13 gecontroleerd in de “while” conditie: “!sr.EndOfStream”. De “while” conditie zorgt ervoor dat er gestopt wordt zodra het einde van de stream is bereikt[20]. Hierdoor kan “sr.ReadLine()” nooit null worden en had er ook geen controle op gedaan hoeven worden.

Vermindering van de betrouwbaarheid

In de implementatie van de unit wordt er rekening mee gehouden dat er problemen kunnen optreden tijdens de uitvoering. Op regel 20-22 en 33-45 is een foutafhandeling geïmplementeerd.

Deze foutafhandeling zorgt ervoor dat er een foutmelding wordt getoond aan de gebruiker van het softwaresysteem. De al ingevoerde samples worden hierna teruggedraaid en het inlezen van de samples wordt afgebroken.

Door het advies van ReSharper is wel voorkomen dat er een NullReferenceException op kan treden, maar invulling van een eventuele foutafhandeling is onbelicht gebleven. In dit source code fragment is de kans dat deze exception optreedt niet aanwezig. Als dit wel mogelijk was geweest, dan was de kans ontstaan dat er verkeerde “samples” werden toegevoegd aan de database Repository van EMS.

Verhoging van de onderhoudbaarheid

Door de toevoeging van het IF statement is duidelijk geworden dat “line” in sommige situaties null kan zijn. Hierdoor weet de ontwikkelaar dat hier een NullReferenceException had kunnen ontstaan en dat dit door het IF statement wordt voorkomen.

Dit kan de ontwikkelaar helpen om een beeld te krijgen van de mogelijke exceptions die kunnen optreden. Hiernaast kan de ontwikkelaar een ELSE statement toevoegen met instructies die van belang zijn als “line” null blijkt te zijn.

Hiervoor geldt wel dat dit IF statement alleen zin heeft als het ook daadwerkelijk kon gebeuren dat “line” null werd, wat in dit source code fragment niet het geval is.

Vermindering van de onderhoudbaarheid

Het toevoegen van het IF statement heeft de onderhoudbaarheid niet noemenswaardig verminderd. De unit is niet veel in CC en LOC toegenomen en ook de analyseerbaarheid van de unit is niet bedreigd door deze aanpassing.

Eindafweging

De controle of “line” niet null is een aanzet naar een verhoging van de betrouwbaarheid van het softwaresysteem. De unit doet foutafhandeling als er een fout optreedt tijdens het inlezen van het CSV bestand.

Het IF statement heeft er niet voor gezorgd dat er ook foutafhandeling wordt uitgevoerd als “line” null is. Om de betrouwbaarheid blijvend te verhogen zou in het ELSE statement instructies geplaatst moeten worden die voor een gepaste foutafhandeling zorgen als “line” null is.

Maar bovenal was het toevoegen van dit IF statement helemaal niet nodig omdat de source code zo is geschreven dat ”line” niet null kan zijn. Hierdoor heeft deze wijziging geen enkele meerwaarde voor de kwaliteitsaspecten van het softwaresysteem.

4.2.2.3 Source code fragment 3

In dit source code fragment (Tabel 6) heeft ReSharper een mogelijke NullReferenceException gevonden in “response” in regel 7 van deze unit. Volgens ReSharper kan het voorkomen dat “response” op die regel in sommige gevallen null is. Om te voorkomen dat de NullReferenceException kan optreden adviseert ReSharper om, voordat met “response” wordt gecommuniceerd, eerst te controleren of “response” niet null is.

Deze aanpassing zorgde ervoor dat ReSharper vervolgens aangaf dat “response.GetResponseStream()” niet meer direct aan de StreamReader “reader” kon worden toegekend. “response.GetResponseStream()” zou kunnen leiden tot een toewijzing van null aan “reader”, dit is volgens de documentatie[21] van StreamReader niet toegestaan.

Het oplossen hiervan heeft geleid tot de 2e wijziging in de source code. Eerst wordt "response.GetResponseStream()" in "responseStream" geladen, daarvan wordt vervolgens gecontroleerd of deze niet null is. Als dat zo is wordt de "responseStream" toegekend aan de StreamReader "reader".

```
1  private XmlDocument GetXml(string Url, string Username, string Password) {
2     try {
3        var results = new XmlDocument();
4        Url = string.Format("{0}{1}", _baseUrl, Url);
5        var request = WebRequest.Create(Url);
6        request.Credentials = new NetworkCredential(Username, Password);
7        var response = request.GetResponse();
8        if (response != null) {
9           Stream responseStream = response.GetResponseStream();
10          if (responseStream != null) {
11             using (var reader = new StreamReader(responseStream)) {
12                var xmlString = reader.ReadToEnd();
13                try {
14                   results.LoadXml(xmlString);
15                } catch {
16                   LogWarning("Received Unexpected Response from Unfuddle Server.");
17                }
18             }
19          }
20       }
21       return results;
22    }
23    catch(Exception ex) {
24       LogWarning(string.Format("Exception occurred: {0}", ex.Message));
25       return null;
26    }
27 }
```

Tabel 6 – ScrewTurn Wiki: ScrewTurn.Wiki.Plugins.PluginPack.GetXml – na aanpassingen

Verhoging van de betrouwbaarheid

Het doorvoeren van deze wijzigingen heeft geen feitelijke verhoging van de betrouwbaarheid opgeleverd. Het ontstaan van een mogelijke NullReferenceException is wel voorkomen, maar hiervan kan niet worden geprofiteerd. Hoe dit komt staat beschreven in het volgende hoofdstuk.

Vermindering van de betrouwbaarheid

De werking van "GetResponse()" zorgt ervoor dat het resultaat "null" wordt als er in "GetResponse()" een fout optreedt. Maar "GetResponse()" geeft hiernaast ook gelijk een exception, hierdoor wordt de source code die volgt, niet meer uitgevoerd.

Met andere woorden, "GetResponse()" zorgt er dus voor dat toegevoegde controle of "response" niet null is, altijd waar is of nooit bereikt zal worden. Door deze toevoeging is dus een mogelijke NullReferenceException voorkomen die nooit kon optreden. De toegevoegde controle werkt hierdoor misleidend en kan een vorm van schijn betrouwbaarheid roepen.

Verhoging van de onderhoudbaarheid

Deze wijzigingen hebben de onderhoudbaarheid van het softwaresysteem niet verbeterd.

Vermindering van de onderhoudbaarheid

Het voorkomen van de mogelijke NullReferenceException heeft ervoor gezorgd dat er 2 IF statements toegevoegd moesten worden. Hiernaast moesten ook extra regels source code worden toegevoegd. Deze wijzigingen hebben de unit zowel in aantal LOC als in CC doen stijgen en hebben op deze manier de analyseerbaarheid negatief beïnvloed en de onderhoudbaarheid verminderd.

De ontwikkelaar zou ten onrechte een ELSE implementatie kunnen schrijven voor een correcte afhandeling als "response" null is. Zoals in de "Vermindering van de betrouwbaarheid" al werd aangegeven zal deze code nooit worden aangesproken.

Eindafweging

Het doorvoeren van de wijzigingen in de source code hebben niet geleid tot een verhoging van de betrouwbaarheid van het softwaresysteem. De onderhoudbaarheid is er ook door afgenomen. Het voorkomen van deze NullReferenceException heeft hiermee geen meerwaarde aan het softwaresysteem toegevoegd.

4.2.2.4 Source code fragment 4

In dit source code fragment (Tabel 7 en Tabel 8) heeft ReSharper het optreden van een mogelijke NullReferenceException voorkomen door de casting aan te passen.

```
1  void LoadValues()
2  {
   /* ... */
27    if (activeItem != null) {
28       activeItem.BackColor = Color.LightGreen;
29       txtActiveValue.Text = (activeItem.Tag as Sample).Value.ToString();
30    } else
31       txtActiveValue.Text = "Geen actieve waarde";
32 }
```

Tabel 7 – EMS: EmsClient.DataEdit.FlowCompareDataEditControl.LoadValues – voor aanpassingen

```
1  void LoadValues()
2  {
3     lstValues.Items.Clear();
4
5     if (compareData.Samplelist == null)
6        return;
7
8     List<Sample> sortedSamples = new List<Sample>(compareData.Samplelist.Samples);
9     sortedSamples.Sort(new SampleComparer());
10
11    ListViewItem activeItem = null;
12    foreach (Sample smp in sortedSamples)
13    {
14       string dateTimeDesc = smp.Sampletime.ToString();
15       if (smp.Sampletime > DateTime.Now)
16          dateTimeDesc += " (Nog niet actief)";
17
18       ListViewItem lvi = new ListViewItem(new string[] {smp.Value.ToString(),
         dateTimeDesc});
19       lvi.Tag = smp;
20       lstValues.Items.Add(lvi);
21
22       if (smp.Sampletime <= DateTime.Now &&
23          (activeItem == null || ((Sample)activeItem.Tag).Sampletime < smp.Sampletime))
24          activeItem = lvi;
25    }
26
27    if (activeItem != null) {
28       activeItem.BackColor = Color.LightGreen;
29       txtActiveValue.Text = ((Sample) activeItem.Tag).Value.ToString();
30    } else
31       txtActiveValue.Text = "Geen actieve waarde";
32 }
```

Tabel 8 – EMS: EmsClient.DataEdit.FlowCompareDataEditControl.LoadValues – na aanpassingen

In Tabel 8 is te zien dat “activeItem.Tag” expliciet gecast wordt naar een object van het type “Sample”. In Tabel 7 is te zien hoe de casting naar “Sample” daarvoor plaats vond door gebruik te maken van “AS” casting.

Verhoging van de betrouwbaarheid

Door het gebruik van “AS” casting kan het gebeuren dat “Sample” na de casting null is. Volgens de documentatie[22] probeert “AS” namelijk een object te casten naar het gewenste type en als dit niet

lukt dan resulteert het in “null”. Door de expliciete casting kan dit niet meer gebeuren en kan “Value” worden aangesproken zonder dat de NullReferenceException kan optreden.

Vermindering van de betrouwbaarheid

Expliciet casten brengt risico’s met zich mee. Als het casten naar “Sample” niet lukt, omdat “activeItem.Tag” niet te converteren is naar “Sample” zal er een InvalidCastException optreden. In deze situatie was het veiliger geweest om eerst “activeItem.Tag” naar “Sample” te casten met “AS”, te controleren of dit gelukt is en daarna “Value” aan te spreken.

Verhoging van de onderhoudbaarheid

Door de minimale aanpassingen is de onderhoudbaarheid niet in positieve zin beïnvloed.

Vermindering van de onderhoudbaarheid

Door de minimale aanpassingen is de onderhoudbaarheid niet in negatieve zin beïnvloed.

Eindafweging

Door de aanpassingen is het optreden van een mogelijke NullReferenceException voorkomen. Maar omdat dit met expliciete casting is gebeurd kan er alsnog een exception optreden. Het was beter geweest als dit was opgelost met “AS” casting en een extra controle. Dat de onderhoudbaarheid dan wel negatief beïnvloed zou worden, door de toename van de CC en het aantal LOC van de unit, weegt niet op tegen de mogelijke exception die hiermee wordt voorkomen.

4.2.3 Analyse

ReSharper heeft aangegeven in welke units van de softwaresystemen exceptions konden optreden. Van deze exceptions heeft ReSharper aangegeven hoe het optreden ervan voorkomen kon worden. De adviezen die ReSharper aandraagt hebben niet altijd de gewenste verhoging in de betrouwbaarheid met zich mee gebracht.

Er zijn situaties voorgekomen waar volgens ReSharper exceptions voorkomen konden worden, maar in de praktijk het niet mogelijk was dat deze exceptions ook echt zouden plaats vinden (source code fragment 2 en 3). Ook zijn er oplossingen voorgesteld om mogelijke exceptions te voorkomen die nieuwe mogelijke exceptions met zich mee brachten (source code fragment 4).

Hiernaast lijkt ReSharper bij het geven van de adviezen geen rekening te houden met de overige instructies en het doel van de unit. Het voornaamste doel lijkt het voorkomen van de mogelijke exception te zijn, zonder rekening te houden welke effecten dit op de betrouwbaarheid en de onderhoudbaarheid kunnen hebben.

Dit is te goed te zien in source code fragment 2. ReSharper heeft hier de toevoeging van het IF statement aanbevolen om zo te controleren of “line” niet null is. De andere instructies in die unit draaien de uitgevoerde mutaties terug op het moment dat er een probleem optreedt. ReSharper heeft dit verder niet opgemerkt en er worden dan ook geen mutaties terug gedraaid als geconstateerd wordt dat “line” null is.

In source code fragment 1 verandert de onderhoudbaarheid in positieve en in negatieve zin door de adviezen van ReSharper. Hier was het mogelijk om de negatieve effecten van de onderhoudbaarheid gedeeltelijk te verminderen. In source code fragment 2 en 4 wordt de onderhoudbaarheid niet veel beïnvloed, in deze gevallen kan de afweging gemaakt worden door te kijken naar de effecten op de betrouwbaarheid. In source code fragment 3 is de onderhoudbaarheid achteruit gegaan, terwijl de betrouwbaarheid ook niet is gestegen.

4.3 Wat is de impact op de kwaliteitsaspecten?

In welke mate worden de kwaliteitsaspecten van het softwaresysteem beïnvloed als geprobeerd wordt om de betrouwbaarheid te verhogen? Hoeveel invloed heeft dit op de onderhoudbaarheid van het softwaresysteem?

Voor het beantwoorden van deze deelvraag zal onderzocht worden hoeveel invloed er is uitgeoefend op de onderhoudbaarheid van de softwaresystemen door de pogingen om de betrouwbaarheid te verhogen. Hierbij is het interessant om te weten of de softwaresystemen hierdoor andere SMM scores halen.

4.3.1 Methode

Voor de onderhoudbaarheid zijn r richtlijnen opgesteld waaraan een softwaresysteem moet voldoen om goed onderhoudbaar te zijn. Deze richtlijnen hebben betrekking op de cyclomatische complexiteit (CC) en het volume in LOC van de units van de softwaresystemen.

4.3.1.1 Complexiteit

Het Software Engineering Institute (SEI)[23] heeft categorieën opgesteld waarmee de risicofactor van een unit kan worden ingedeeld. Deze categorieën worden bepaald door de CC score van de unit.

CC	Risicocategorie	Benaming in SMM
1-10	Eenvoudig, zonder veel risico	Laag
11-20	Gemiddeld risico	Gemiddeld
21-50	Complex, groot risico	Hoog
> 50	Onstabiel, heel groot risico	Erg hoog

Tabel 9 – Risicocategorieën volgens SEI

Het SMM heeft op basis van deze risicocategorieën richtlijnen opgesteld waarmee de analyseerbaarheid en testbaarheid van een softwaresysteem in een score uitgedrukt kan worden. Deze score wordt berekend aan de hand van de verdeling van het relatief aantal LOC per risicocategorie van een softwaresysteem.

Score	Gemiddeld	Hoog	Erg hoog
++	25%	0%	0%
+	30%	5%	0%
o	40%	10%	0%
-	50%	15%	5%
--	-	-	-

Tabel 10 – Risicoscore volgens SMM

Vooraf zal van beide softwaresystemen de risicoscore worden berekend. Deze scores zullen opnieuw worden berekend na de aanpassingen en zullen vervolgens met elkaar worden vergeleken. Door te vergelijken of de softwaresystemen van risicoscore veranderen kan bepaald worden hoeveel impact het verhogen van de betrouwbaarheid op de onderhoudbaarheid van de softwaresystemen heeft uitgeoefend.

4.3.1.2 Volume

In het SMM wordt een score berekend waarmee de grootte van het softwaresysteem qua onderhoudbaarheid wordt berekend. Het bepalen van de toegestane volumes voor de verschillende reeksen verschilt per programmeertaal. Voor het opstellen van deze reeksen maakt het SMM gebruik

van een tabel[24] die opgesteld is door Software Productivity Research LLC. Voor de meting in dit onderzoek zal de volumescore (Tabel 11) voor softwaresystemen geschreven in Java worden gebruikt.

Score	Manjaren	LOC (x1000)
++	0 – 8	0 – 66
+	8 – 30	66 – 246
o	30 – 80	246 – 665
-	80 – 160	655 – 1.310
--	> 160	> 1.310

Tabel 11 – Volumescore voor Java softwaresystemen volgens Software Productivity Research LLC

De softwaresystemen in dit onderzoek zijn geschreven in .NET. Omdat zowel Java als .NET objectgeoriënteerde programmeertalen zijn kan de volumescore voor Java gebruikt worden voor het berekenen van de scores.

Vooraf zal van beide softwaresystemen de volumescore worden berekend. Deze scores zullen opnieuw worden berekend na de aanpassingen en zullen vervolgens met elkaar worden vergeleken. Door te vergelijken of de softwaresystemen van risicoscore veranderen kan bepaald worden hoeveel impact het verhogen van de betrouwbaarheid op de onderhoudbaarheid van de softwaresystemen heeft uitgeoefend.

4.3.2 Resultaten

Tabel 12 geeft de verdeling van de units per complexiteitschaal van ScrewTurn Wiki weer. Per schaal staat weergegeven hoeveel units zich voor en na de aanpassingen in deze schaal bevinden.

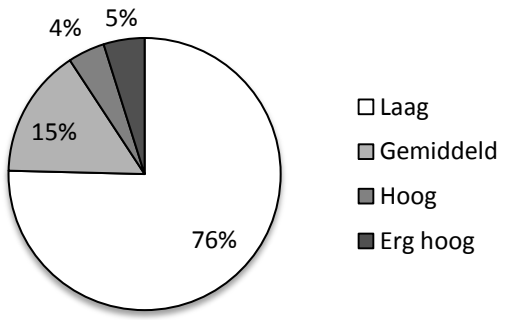
CC	Aantal units voor aanpassing	Aantal units na aanpassing	Verschil
1-10	3751	3744	-7
11-20	108	112	+4
21-50	20	23	+3
> 50	6	6	+0

Tabel 12 – ScrewTurn Wiki: Verplaatsing van units per CC schaal

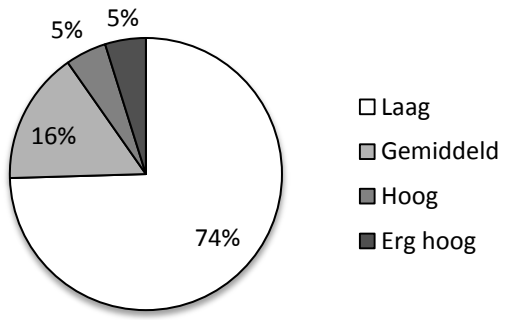
CC	Aantal units voor aanpassing	Aantal units na aanpassing	Verschil
1-10	819	819	+0
11-20	1	1	+0
21-50	0	0	+0
> 50	0	0	+0

Tabel 13 – EMS: Verplaatsing van units per CC schaal

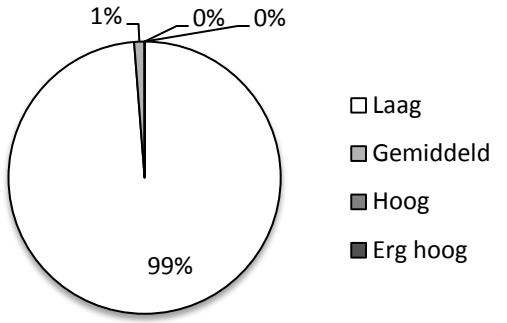
Om tot een berekening van een risicoscore te kunnen komen is de percentuele verdeling van het aantal LOC verdeeld over de CC risicocategorieën nodig. Deze verdeling wordt getoond in Figuur 10 - Figuur 13.



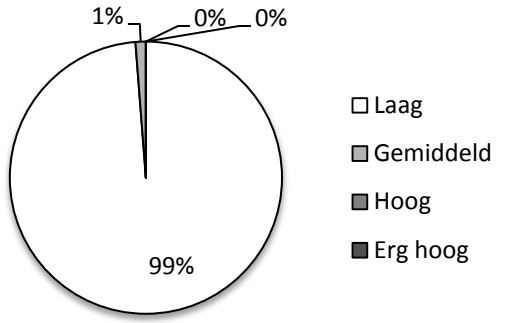
Figuur 10 – ScrewTurn Wiki: Verdeling LOC per CC niveau – voor aanpassingen



Figuur 11 – ScrewTurn Wiki: Verdeling LOC per CC niveau - na aanpassingen



Figuur 12 – EMS: Verdeling LOC per CC niveau – voor aanpassingen



Figuur 13 – EMS: Verdeling LOC per CC niveau - na aanpassingen

Door het aanpassen van ScrewTurn Wiki heeft er een verschuiving plaats gevonden in de verdeling van het aantal LOC per CC. Het percentage LOC met een lage CC is afgenomen en de percentages van LOC met een gemiddelde en hoge CC zijn gestegen. ScrewTurn Wiki scoorde volgens het SMM voor de aanpassing "-" en na de aanpassing is deze score onveranderd gebleven. Het aanpassen van EMS had geen zichtbaar effect op de verdeling van de LOC per CC niveau. EMS scoorde zowel voor als na de aanpassing "++".

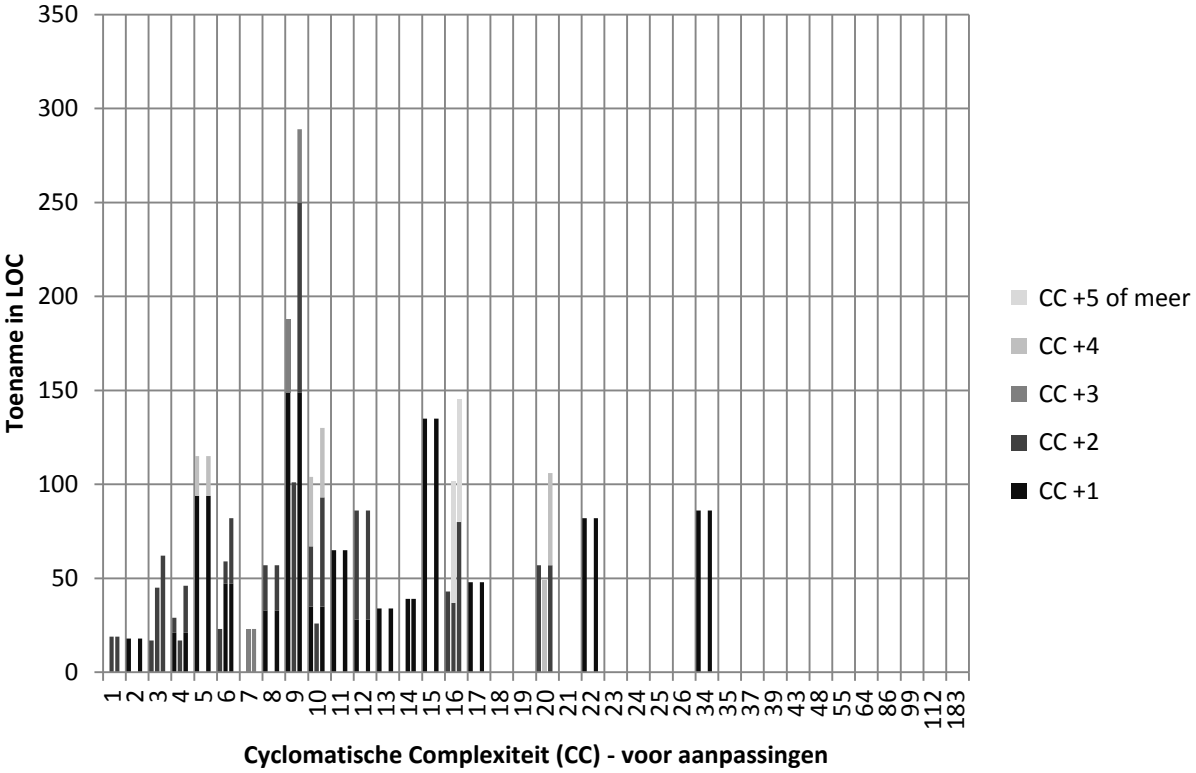
Door het aanpassen van de softwaresystemen is ook het aantal LOC ervan toegenomen, dit is te zien in Tabel 14.

Softwaresysteem	LOC voor aanpassingen	LOC na aanpassingen	Verschil
ScrewTurn Wiki	32899	33043	+ 144
EMS	3567	3574	+ 7

Tabel 14 – LOC volume van de softwaresystemen

Beide softwaresystemen scoorden voor de aanpassing een "++" score volgens het SMM. Na de aanpassingen was de score nog steeds "++".

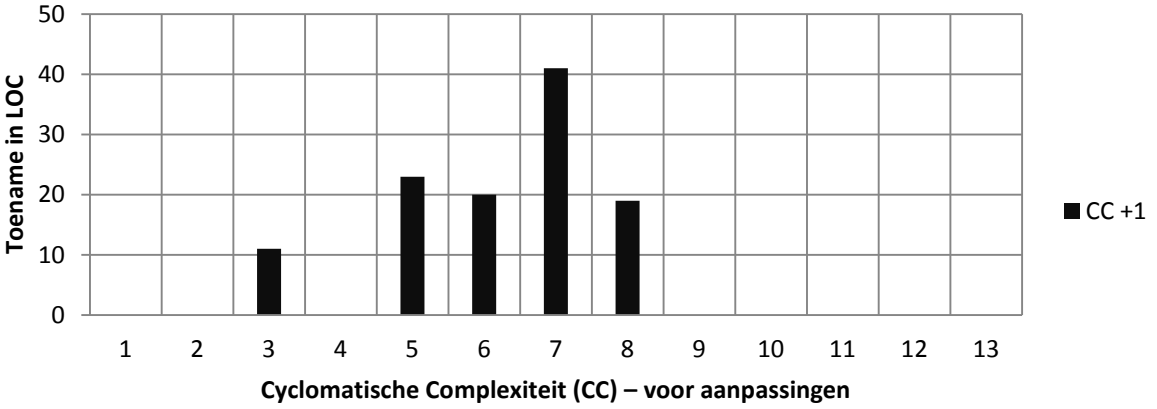
De daadwerkelijke verschuiving van de LOC per CC van de beide softwaresystemen zijn in Figuur 14 en Figuur 15 weergeven. Per CC wordt getoond hoeveel LOC met hoeveel CC zijn gestegen.



Figuur 14 – ScrewTurn Wiki: Toename van Lines of Code (LOC) per Cyclomatische Complexiteit (CC)

In Figuur 14 zijn per CC 3 staven getoond, per staaf is de groei in aantal LOC per CC weergegeven. De staven zijn voorzien van meerdere kleuren, deze kleuren laten zien hoeveel LOC met hoeveel CC zijn gestegen.

De eerste staaf toont de toename in LOC tijdens de eerste aanpassingsronde, dit is de ronde waar ReSharper advies kon geven hoe de mogelijke exceptions voorkomen worden. De tweede staaf toont de toename van de tweede ronde, de ronde waar ReSharper geen advies kon geven waarmee de exception voorkomen kon worden. De laatste staaf toont de som van beide aanpassingen.



Figuur 15 – EMS: Toename van Lines of Code (LOC) per Cyclomatische Complexiteit (CC)

Bij EMS konden alle mogelijke exceptions in de eerste aanpassingsronde worden opgelost. In bijna alle gevallen zorgde dit voor een toename van 1 stap in CC. De resultaten zijn in Figuur 15 weergeven. Bij ScrewTurn Wiki bevindt de meeste toename zich in de units die een CC hebben tussen de 1 – 10 en 11 – 20. De overige aanpassingen bevinden zich in de groep 21 – 50.

Om een duidelijk beeld van te krijgen met hoeveel LOC de units zijn toegenomen is Tabel 15 toegevoegd. In deze tabel staan de resultaten uit Figuur 14 als gegevenstabel toegevoegd.

Toename in CC	1e bewerking		2e bewerking		1e + 2e bewerking	
+1	828 LOC	69,8 %	86 LOC	17,9 %	914 LOC	54,8 %
+2	262 LOC	22,0 %	257 LOC	53,5 %	519 LOC	31,1 %
+3	39 LOC	3,3 %	23 LOC	4,8 %	62 LOC	3,7 %
+4	58 LOC	4,9 %	49 LOC	10,2 %	107 LOC	6,4 %
Meer dan 4	0 LOC	0,0 %	65 LOC	13,5 %	65 LOC	3,8 %
Totaal	**1187 LOC**	**100,0 %**	**480 LOC**	**100,0 %**	**1667 LOC**	**100,0 %**

Tabel 15 – ScrewTurn Wiki: Toename in LOC per CC

Voor EMS is deze tabel niet toegevoegd omdat in Figuur 15 al is te zien waar en met welk volume de verschuivingen hebben plaats gevonden.

4.3.3 Analyse

De resultaten geven aan dat de softwaresystemen niet in score veranderd zijn volgens het SMM. Bij ScrewTurn Wiki is de CC verdeling wel in negatieve zin veranderd. Het percentage LOC met een laag CC is afgenomen (2 procent). Deze afname is terug te vinden als toename in de groepen LOC met een gemiddeld en hoog CC. Deze 2 groepen zijn allebei met 1 procent gestegen.

Omdat er voor de aanpassingen al LOC waren met een erg hoge CC was de score volgens het SMM voor de aanpassing "-". De verschuivingen hebben hierdoor geen effect gehad op de score. Bij EMS was er een dusdanig kleine verschuiving dat dit bijna niet terug te zien was in de resultaten. De verschuiving had dan ook geen effect op de score volgens het SMM.

De toename in aantal LOC hebben ook niet geleid tot grote verschuivingen. De scores van het SMM zijn ook op dat gebied niet veranderd. De verschuiving was groter geweest als er source code was toegevoegd, met instructies die uitgevoerd zouden worden, als de mogelijke exception was onderschept.

De verschillen (voor en na de aanpassingen) zijn zo klein dat het twijfelachtig is of het terecht zou zijn geweest als hierdoor wel de score binnen het SMM zou zijn veranderd. De score zou dan zijn veranderd omdat het een drempelwaarde zou hebben overgeschreven, maar niet omdat er een grote verschuiving plaats gevonden heeft. De verschuiving is in Tabel 16 weergegeven.

Softwaresysteem	Verschuiving in CC				Verschuiving in LOC
	Laag	Gemiddeld	Hoog	Erg hoog	
ScrewTurn Wiki	– 0,86 %	+ 0,38 %	+ 0,50 %	– 0,21 %	+ 0,43 %
EMS	± 0,00 %	± 0,00%	± 0,00 %	± 0,00 %	+ 0,19 %

Tabel 16 – Verschuiving in CC en LOC binnen beide softwaresystemen

5 Analyse

In dit onderzoek is bij 2 softwaresystemen geprobeerd om de betrouwbaarheid te verhogen. Met behulp van ReSharper zijn de mogelijke exceptions opgezocht en waar mogelijk is de voorgestelde oplossing van ReSharper geïmplementeerd. Waar ReSharper geen oplossing kon aandragen is zelf een oplossing toegevoegd om er voor te zorgen dat de exception niet meer kon optreden.

Bij ScrewTurn Wiki zijn er 82 exceptions verdeeld over 54 units (1,38% van alle units) opgespoord en opgelost. Bij EMS waren er 30 mogelijke exceptions gevonden, hiervan zijn 7 mogelijke exceptions die verdeeld waren over 6 units (0,73% van alle units) opgelost.

De overige 23 mogelijke exceptions die konden optreden bevonden zich in gegenereerde source code. De oorsprong van deze mogelijke exceptions bevindt zich niet op de locatie waar ze zijn aangetroffen. Om die reden zijn deze exceptions buiten beschouwing gelaten tijdens dit onderzoek.

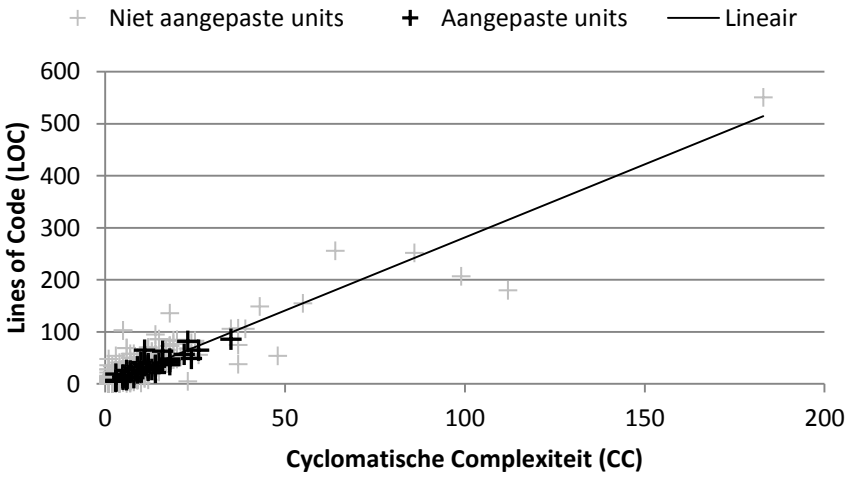
Tijdens het observeren van de wijzigingen die zijn doorgevoerd op advies van ReSharper is te zien dat de doorgevoerde wijzigingen niet altijd de beoogde verhoging van de betrouwbaarheid met zich mee hebben gebracht. ReSharper heeft bij bepaalde source code gemeld dat er een mogelijke exception zou kunnen optreden terwijl die exception op die plek niet kon optreden.

Deze mogelijke exception werd dan al voorkomen door omliggende source code. Bij de gevallen waar wel een exception kon optreden leken de adviezen van ReSharper vooral bedoeld om de exception zo ‘snel’ mogelijk te voorkomen. De adviezen van ReSharper leken geen rekening te houden met, of de voorgestelde oplossing op het gebied van onderhoudbaarheid, ook de beste oplossing was.

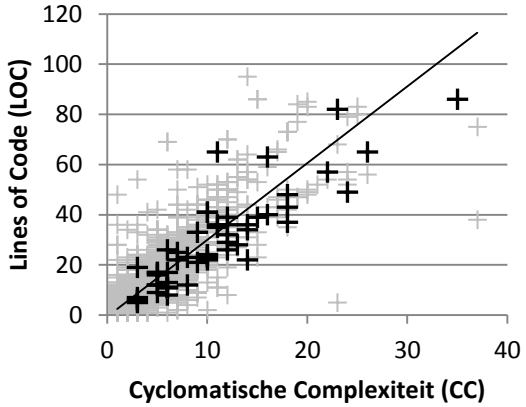
Door het wegnemen van de mogelijke exceptions zijn de onderhoudbaarheidskarakteristieken van de units waarin deze exceptions zich bevonden veranderd. In bijna alle gevallen zijn van deze units zowel de CC als het aantal LOC gestegen. De toename van het aantal LOC van de units heeft ook geleid tot een toename van het totaal aantal LOC van de softwaresystemen.

In Figuur 16, Figuur 17 en Figuur 18 is te zien welke units door de aanpassingen zijn veranderd. Op de X-as is de CC van de units getoond en op de Y-as wordt het aantal LOC van de units getoond. In de grafieken is te zien dat het merendeel van de aangepaste units een hoger CC hebben ook het aantal LOC hoger is.

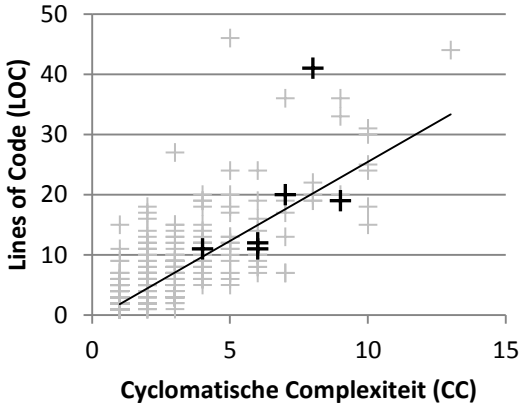
Dit laat zich deels verklaren omdat het ontstaan van een hoger CC afhankelijk is van de aanwezigheid van LOC. De CC kan alleen maar toenemen als het aantal LOC ook toeneemt. Hiernaast zijn de units met veel LOC en een lage CC minder complex waardoor de kans op mogelijke exceptions lager kan liggen. De units met een hoog CC en een laag LOC kunnen units zijn waar een case switch of veel controles in zijn ondergebracht. Dergelijke units sturen vaak andere units aan en zijn daarnaast minder operationeel. Hierdoor is de kans kleiner dat ze source code bevatten waar mogelijke exceptions kunnen optreden.



Figuur 16 – ScrewTurn Wiki: Aangepaste units



Figuur 17 – ScrewTurn Wiki: Aangepaste units (ingezoomd)



Figuur 18 – EMS: Aangepaste units

Tijdens het beantwoorden van de deelvragen is gekeken welke invloed er op de onderhoudbaarheid is uitgeoefend door de wijzigingen aan de softwaresystemen. De resultaten hebben laten zien dat de onderhoudbaarheid licht is afgenomen door de wijzigingen.

Voor het vergelijken van de onderhoudbaarheid van voor en na de aanpassingen is gebruikt gemaakt van de scores zoals die berekend worden in het SMM. Deze scores volgens het SMM waren na de aanpassingen niet veranderd ten opzichte van de scores voor de aanpassingen.

Hierdoor zou je kunnen stellen dat door het wegnemen van de mogelijke exceptions, die ook daadwerkelijk de betrouwbaarheid van de softwaresystemen verhogen, de hypothese “H1: Door het verhogen van de betrouwbaarheid wordt de onderhoudbaarheid slechter.” niet waar is. De alternatieve hypothese: “AH1.1 Door het verhogen van de betrouwbaarheid blijft de onderhoudbaarheid gelijk.” zou dan wel correct zijn.

Dit neemt echter niet weg dat ook al is de onderhoudbaarheid volgens de scores van het SMM niet afgenomen, er wel degelijk een afname van de is onderhoudbaarheid is gemeten. Deze verandering in onderhoudbaarheid is ook zichtbaar geworden tijdens het beantwoorden van de deelvraag: “S-RQ3: Wat is de impact van een mogelijke afhankelijkheid op de kwaliteitsaspecten?”

Het SMM hanteert reeksen voor het bepalen van de scores van de onderhoudbaarheidskarakteristieken. Hierdoor blijft de score volgens het SMM hetzelfde als de verandering van de onderhoudbaarheid karakteristiek binnen de reeks blijft. Dit was bij de in het onderzoek gebruikte softwaresystemen dan ook het geval.

Omdat door het verhogen van de betrouwbaarheid de onderhoudbaarheid toch is verminderd, kan de hypothese: “H1: Door het verhogen van de betrouwbaarheid wordt de onderhoudbaarheid slechter.” alsnog als correct worden aangenomen.

6 Conclusie

In dit onderzoek zijn bij 2 softwaresystemen de effecten op de onderhoudbaarheid onderzocht als met ReSharper geprobeerd werd om de betrouwbaarheid van deze softwaresystemen te verhogen. Dit is gedaan door met ReSharper in 2 softwaresystemen te zoeken naar mogelijke exceptions die konden optreden. Deze mogelijke exceptions zijn weggenomen, dit is zoveel mogelijk gedaan door de adviezen van ReSharper uit te voeren. In de gevallen dat deze niet toereikend waren, zijn de mogelijke exceptions weggenomen door handmatig instructies en controles toe te voegen.

De resultaten hebben laten zien dat er hierdoor een verplaatsing heeft plaats gevonden in de onderhoudbaarheid van de software. Deze verschuiving is in kaart gebracht en hierbij is gekeken welke invloed ze hebben uitgeoefend op de onderhoudbaarheidskarakteristieken die in het SMM worden gebruikt. Hier is uit naar voren gekomen dat de invloeden op de onderhoudbaarheid erg klein waren. De invloed was zo klein dat de softwaresystemen voor en na de aanpassingen volgens het SMM niet in onderhoudbaarheid waren veranderd en dezelfde scores behaalden.

Hiernaast is ook waargenomen dat de adviezen die ReSharper doet niet altijd de betrouwbaarheid hebben verbeterd. Zo zijn er gevallen waargenomen waarbij er helemaal geen exception kon optreden maar ReSharper ze wel gevonden dacht te hebben.

Als een gevonden exception echt kon plaats vinden, is de oplossing die ReSharper voorstelt, vooral gericht op het voorkomen van de exception. Hierbij werd het doel van de unit niet in beschouwing genomen en kon dit leiden tot situaties waar de betrouwbaarheid niet altijd werd verbeterd. Hiernaast leidde dit er ook toe dat sommige adviezen van ReSharper meer negatieve invloed uitoefende op de onderhoudbaarheid dan nodig was om de exception te voorkomen.

De analyses hebben laten zien dat ReSharper goed te gebruiken is om naar mogelijke exceptions in softwaresystemen te zoeken. Voor het toetsen en eventueel voorkomen van de mogelijke exceptions lijkt ReSharper niet geschikt. Van de mogelijke exceptions dient zelf onderzocht te worden of ze daadwerkelijk kunnen optreden. Hiernaast dient ook het advies van ReSharper onderzocht te worden om vast te stellen of de betrouwbaarheid er daadwerkelijk wordt verhoogd. Van het advies dient ook onderzocht te worden welke invloed het heeft op de onderhoudbaarheid en of de verdere werking van het softwaresysteem gewaarborgd blijven.

De behoefte om mogelijke exceptions in een softwaresysteem te willen voorkomen, kan per situatie verschillen. Zo kan het bij sommige softwaresystemen juist wel wenselijk zijn als een mogelijke exception niet wordt voorkomen. De functie die het softwaresysteem moet vervullen kan hier een rol in spelen.

Wanneer de exception zich kan voordoen in een “class library” of een Application Programming Interface (API), dan kan het een overweging zijn om de exception juist wel te laten optreden. Hierdoor weet de ontwikkelaar die de class library op API implementeert dat er een probleem is opgetreden en kan de ontwikkelaar zelf bepalen hoe er met de exception omgegaan dient te worden.

Bij volledig geautomatiseerde softwaresystemen kan het van belang zijn dat er geen exceptions kunnen optreden. Dit belang kan komen omdat dergelijke softwaresystemen niet manueel bediend worden. Hierbij kan het gebeuren dat niet direct wordt opgemerkt als het softwaresysteem niet meer correct functioneert. Dit soort softwaresystemen kunnen uitgerust zijn met een mechanisme dat de correcte werking van het softwaresysteem bewaakt. Als de exception hierin optreedt, bestaat de kans dat de signalering niet meer werkt en mogelijke problemen in de

werking niet worden opgemerkt. Dit soort geautomatiseerde softwaresystemen kunnen bijvoorbeeld softwaresystemen zijn die de aansturing van bruggen en seinen verzorgen.

Ook bij softwaresystemen die vitale functies moet bewaken of ondersteunen is het van het grootste belang dat het softwaresysteem te allen tijde blijft functioneren. Het optreden van een exception zou deze waarborging in gevaar kunnen brengen. Dit zijn softwaresystemen die bijvoorbeeld de veiligheid van mensen of dieren moet waarborgen of bewaken.

6.1 Discussie over de validiteit

Deze paragraaf beschrijft welke aspecten uit het onderzoek de getrokken conclusies zouden kunnen beïnvloeden.

6.1.1 Kwaliteit van de adviezen van ReSharper

In het onderzoek zijn de adviezen van ReSharper aangenomen om de betrouwbaarheid van de softwaresystemen te verhogen. Het onderzoek naar de kwaliteit van deze adviezen hebben naar voren gebracht dat ze niet in alle gevallen de betrouwbaarheid van de softwaresystemen ook echt hebben verhoogd. Hierdoor zijn de effecten op de onderhoudbaarheid die zijn gemeten, niet altijd het gevolg van verhoging van de betrouwbaarheid en kunnen op deze manier de resultaten zijn beïnvloed.

Hiernaast lijken de adviezen van ReSharper als primair doel het voorkomen van de exceptions na te streven. Hierdoor zijn er situaties voorgekomen waar eigenlijk meer wijzigingen aan de source code nodig waren om de beoogde verbetering van de betrouwbaarheid te realiseren. Door het niet realiseren van deze wijzigingen, zijn de eventuele effecten ervan op de onderhoudbaarheid niet gemeten.

6.1.2 Betrouwbaarheid van cyclomatische complexiteit (CC)

In recent onderzoek [25] wordt de aandacht gevestigd op de betrouwbaarheid voor het gebruiken van CC als metriek voor het meten van onderhoudbaarheid. In hun resultaten komt naar voren dat er aanwijzingen zijn die er op duiden dat CC en analyseerbaarheid van source code niet zo aan elkaar gerelateerd zijn als altijd werd gedacht. Aangezien in dit onderzoek ook CC gebruikt is om onderhoudbaarheid te meten, kan het de resultaten uit dit onderzoek beïnvloeden als er uitgegaan wordt van de bevindingen uit dit nieuwe onderzoek.

6.1.3 Beperkte hoeveelheid softwaresystemen

Aangezien in het onderzoek twee softwaresystemen zijn onderzocht kunnen er geen conclusies worden getrokken waarvan met grote zekerheid gesteld kan worden dat ze voor alle softwaresystemen gelden. Het was helaas niet mogelijk om de source code waar de metingen op verricht werden volledig geautomatiseerd aan te kunnen passen. Dit kwam doordat de softwaresystemen gedeeltelijk met de hand in betrouwbaarheid verhoogd moesten worden.

6.1.4 Dezelfde programmeertaal van de softwaresystemen

De softwaresystemen die in dit onderzoek zijn gebruikt zijn allebei geschreven in .NET., Hierdoor bestaat de mogelijkheid dat de uitkomsten van het onderzoek afwijken als er softwaresystemen worden toegevoegd aan de dataset die geschreven zijn in andere programmeertalen.

6.2 Toekomstig werk

In deze paragraaf staan voorstellen voor vervolgonderzoek genoemd die uitgevoerd zouden kunnen worden op basis van dit onderzoek.

6.2.1 Cyclomatische complexiteit (CC) vervangen

Om de betrouwbaarheid van CC, als metriek voor het meten van onderhoudbaarheid vast te stellen, zou dit onderzoek herhaald kunnen worden met de alternatieve metrieken die gepresenteerd worden in het onderzoek van Vinju en Godfrey[25]. In dit onderzoek worden alternatieve metrieken geïntroduceerd om de analyseerbaarheid van source code beter te kunnen bepalen. Door dit onderzoek te herhalen met deze nieuwe metrieken kan worden vastgesteld of de conclusies van dit onderzoek afwijken of dat dit geen verschil maakt.

6.2.2 Softwaresystemen toevoegen

Om tot een meer generaliserende uitspraak te kunnen komen of er een relatie bestaat tussen betrouwbaarheid en onderhoudbaarheid zou dit onderzoek herhaald kunnen worden op meer softwaresystemen. Door softwaresystemen die geschreven zijn in andere programmeertalen toe te voegen, kan onderzocht worden of de conclusies taalafhankelijk zijn.

Referenties

[1] IEEE, „Std. 610.12 "IEEE Standard Glossary of Software Engineering Terminology",” 1990.

[2] B. W. Boehm, „Software Engineering Economics,” IEEE, 1984.

[3] B. P. Lientz, „Issues in Software Maintenance,” Computing Surveys, 1983.

[4] B. P. Lientz, E. B. Swanson en G. E. Tompkins, „Characteristics of application software maintenance,” *Comm of ACM, pp: 461-471,* nr. no. 6, Jun. 1978.

[5] ISO, „ISO/IEC 9126-1: Software engineering -- Product quality -- Part 1: Quality model,” 2001.

[6] A. Zeller, Why programs fail; A guide to systematic debugging; Second edition, 2009.

[7] „ScrewTurn Wiki,” [Online]. Available: http://www.screwturn.eu.

[8] „ReSharper,” [Online]. Available: http://www.jetbrains.com/resharper.

[9] I. Heitlager, T. Kuipers en J. Visser, „A Practical Model for Measuring Maintainability,” Software Improvement Group, 2007.

[10] „Software Improvement Group (SIG),” [Online]. Available: http://www.sig.eu/nl/.

[11] T. McCabe, „A complexity measure,” IEEE, 1976.

[12] B. Luijten, „The Influence of Software Maintainability on Issue Handling,” 2009.

[13] D. Bijlsma, „Indicators of Issue Handling Efficiency,” 2010.

[14] V. Okun, W. F. Guthrie, R. Gaucher en P. E. Black, „Effect of Static Analysis Tools on Software Security: Preliminary Investigation,” ACM, 2007.

[15] P. J. Guo en D. Engler, „Linux Kernel Developer Responses to Static Analysis Bug Reports,” ACM, 2009.

[16] „Coverity,” [Online]. Available: http://www.coverity.com.

[17] N. Nagappan en T. Ball, „Use of Relative Code Churn Measures to Predict System Defect Density,” ACM, 2005.

[18] M. Glukhikh, M. Moiseev, A. Karpenko en H. Richter, „Software Reliability Estimation Based on Static Error Detection,” IEEE, 2011.

[19] M. Fowler, Refactoring. Improving the design of existing code, 1999.

[20] Microsoft, „MSDN Library - .NET Framework 2.0 - StreamReader.EndOfStream,” [Online]. Available: http://msdn.microsoft.com/en-us/library/system.io.streamreader.endofstream(v=vs.80).

[21] Microsoft, „MSDN Library - .NET Framework 3.5 - StreamReader Constructor,” [Online]. Available: http://msdn.microsoft.com/en-us/library/system.io.streamreader.streamreader(v=vs.90).

[22] Microsoft, „MSDN Library - as (C# Reference),” [Online]. Available: http://msdn.microsoft.com/en-us/library/cscsdfbt(v=vs.100).aspx.

[23] M. Bray en M. Lockheed, Software Technology Reference Guide, Software Engineering Institute, 1997.

[24] Software Productivity Research LLC, „Programming Languages Table, PLT2007c,” 2007.

[25] J. Vinju en M. Godfrey, „What Does Control Flow Really Look Like? Eyeballing the Cyclomatic Complexity Metric,” 2012.

Overzicht van gebruikte figuren

Overzicht van gebruikte tabellen