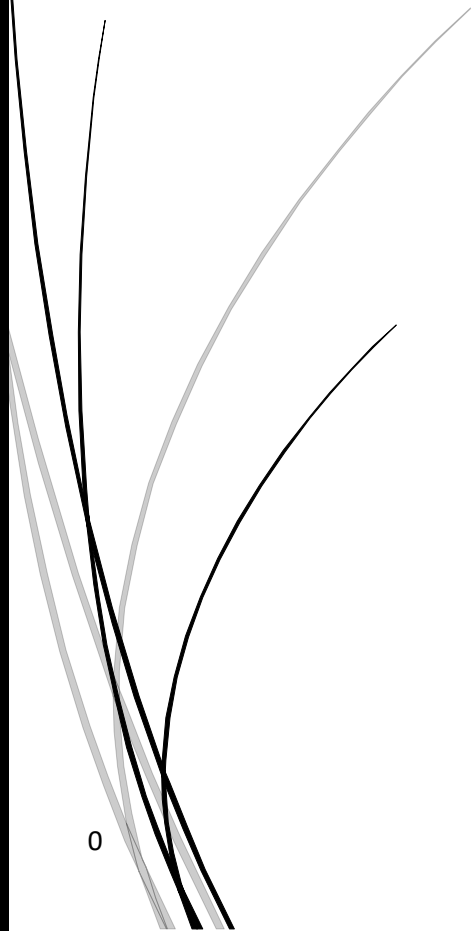
25-9-2016

Onderzoeksplan

IPSENH



RZN
REPOSITORY ZONDER NAAM

Versiebeheer

Versie	Datum	Omschrijving	
0.1	13-09-2016	Initiële versie	Anton
0.2	14-09-2016	Aanzet probleemstelling	Anton
0.2	14-09-2016	Aanleiding	Laurens
0.3	15-09-2016	Aanleiding	Anton
0.4	15-09-2016	Probleemstelling	Anton, Sander
0.5	18-09-2016	Achtergrond over codekwaliteit	Anton
0.6	20-09-2016	Achtergrond over meten codekwaliteit	Sander
0.7	20-09-2016	Achtergrond over ontwikkelstraat	Zairon
0.8	20-09-2016	Achtergrond over toegevoegde waarde ontwikkelstraat	Laurens
1.0	21-09-2016	Afronding voor eerste oplevering	Laurens
1.1.1	22-09-2016	Achtergrond over ontwikkelstraat verbeterd	Zairon, Anton
1.2	23-09-2016	Feedback van Koen verwerkt	Sander
1.2.1	23-09-2016	Controle en kleine taalverbeteringen	Anton
1.3	25-09-2016	Feedback verwerkt	Zairon
2.0	25-09-2016	Kleine verbeteringen en afronding	Sander

Inhoud

Aanleiding 4
Beschrijving van de opdrachtgever 4
Bedrijfsinformatie 4
Opdrachtnemer 4
Aanleiding tot de opdracht 4
Probleemstelling 5
Aanleiding 5
Probleemstelling 5
Onderzoeksvraag 5
Deelvraag 1: Wat verstaat men onder codekwaliteit? 5
Deelvraag 2: Hoe meet je codekwaliteit? 5
Deelvraag 3: Uit welke onderdelen bestaat een ontwikkelstraat? 5
Deelvraag 4: Hoe kan je een ontwikkelstraat inrichten? 5
Deelvraag 5: Hoe kan je een ontwikkelstraat gebruiken om de codekwaliteit te verhogen? 5
Toelichting van de probleemstelling 5
Doelstelling van het onderzoek 5
Wat wil je met het onderzoek bereiken 5
Afbakening van de onderzoeksdoelstelling 5
Verantwoording 6
De reden voor het onderzoek 6
Nieuwswaarde 6
Theoretische relevantie 6
Praktische relevantie 6
Maatschappelijke relevantie 6
Reikwijdte en informatiegehalte 6
Kan de centrale onderzoeksvraag beantwoord worden? 6
Doelgroep 6
Achtergrond 7
Achtergrond theorieën tot probleemstelling 7
Kennisgebieden 7
Inperking kennisgebied 7
Belangrijkste termen en concepten 8
Wat is volgens de literatuur codekwaliteit? 8
Hoe meet je code kwaliteit? 9
Hoe kan je een ontwikkelstraat gebruiken om de codekwaliteit te verhogen? 11

Domeinen 11
Onderzoek structuur en methode 12
Planning 12
Bibliografie 13

Aanleiding

Beschrijving van de opdrachtgever

De opdrachtgever van dit onderzoek is de Hogeschool Leiden. De opdrachtgever heeft verscheidene projecten ter beschikking gesteld voor een vijftal groepen studenten, welke worden gegeven in samenwerking met een derde partij. De hogeschool leiden is de hogeschool waar de studenten lessen volgen. Zij zijn uiteindelijk de partij waar de studenten zich aan verantwoorden.

Bedrijfsinformatie

De Hogeschool Leiden is een instelling voor hoger beroepsonderwijs gevestigd in Leiden. De bedrijfstak die relevant is voor dit onderzoek is de bacheloropleiding Informatica, specialisatie Software Engineering. Deze specialisatie richt zich op het onderwijzen van het ICT-vakgebied doormiddel van onderzoeken en aanverwante opdrachten. De hogeschool leiden stimuleert studenten om zelfstandig en professioneel te worden.

Opdrachtnemer

Dit onderzoek wordt gedaan door drie studenten van de specialisatie Software Engineering van de hogeschool leiden. De naam van het team is “Repository Zonder Naam”. De studenten zijn uit jaar 3 en hoger. IPSENH is het laatste schoolproject waar de studenten aan werken.

Aanleiding tot de opdracht

We gaan voor het schoolproject IPSENH-onderzoek doen naar een ontwikkelstraat en codekwaliteit. We willen dit onderzoek doen om het effect van een ontwikkelstraat te testen, en te testen of dit de codekwaliteit verbetert. Hierbij leren we over codekwaliteit en over ontwikkelstraten, maar het belangrijkste van deze opdracht is het leren onderzoeken en de resultaten hiervan goed te verwerken en te documenteren, om er vervolgens een goed onderbouwd en betrouwbaar advies te kunnen geven.

Probleemstelling

Aanleiding

Wij hebben meerdere keren projecten uitgevoerd waarbij we vanaf niets een softwareapplicatie moesten opleveren. Hier werd er van ons verwacht dat wij ook getest hadden en dat de codekwaliteit hoog was. Echter werd tijdens de projecten niet erg veel aandacht besteedt hieraan, en ook niet aan de regressie van code. Dit komt door onder andere tijdsdruk. Er wordt veel gevraagd in relatief weinig tijd. Dit is omdat software ontwikkelen duur is. Hierdoor liepen we vaak na veranderingen tegen problemen aan, die wij niet voorzien hadden. Bij toekomstige projecten willen we dit voorkomen door meer focus te leggen op testen en continue integreren van nieuwe onderdelen. We denken dat een ontwikkelstraat hierbij kan helpen.

Probleemstelling

De kwaliteit van code is te laag en het kost veel tijd om de kwaliteit hoger te krijgen.

Onderzoeksvraag

Hoe kunnen we in een softwareproject met behulp van een ontwikkelstraat een hogere codekwaliteit realiseren ten op zichten van een softwareproject zonder ontwikkelstraat?

Deelvraag 1: Wat verstaat men onder codekwaliteit?

Deelvraag 2: Hoe meet je codekwaliteit?

Deelvraag 3: Uit welke onderdelen bestaat een ontwikkelstraat?

Deelvraag 4: Hoe kan je een ontwikkelstraat inrichten?

Deelvraag 5: Hoe kan je een ontwikkelstraat gebruiken om de codekwaliteit te verhogen?

Toelichting van de probleemstelling

Deze probleemstelling is gevormd aan de hand van de aanleiding van het onderzoek. Op dit moment vinden wij dat wij tijdens softwareprojecten een te lage codekwaliteit hebben.

Doelstelling van het onderzoek

De codekwaliteit en de efficiëntie van het maken van software, met een hoge codekwaliteit, verhogen door gebruik te maken van een ontwikkelstraat, zonder daar meer tijd voor te gebruiken ten opzichte van een project zonder een ontwikkelstraat.

Wat wil je met het onderzoek bereiken

Wat wij willen bereiken met het onderzoek is om duidelijk te maken wat codekwaliteit is, wat een ontwikkelstraat is en hoe de ontwikkelstraat de codekwaliteit beïnvloedt. We willen weten waarom professionals een ontwikkelstraat gebruiken. Wij willen met dit onderzoek erachter komen hoe wij met de hulp van een ontwikkelstraat een hogere kwaliteit code kunnen opleveren.

Afbakening van de onderzoeksdoelstelling

We relateren het onderzoek aan dit project en richten en onderzoeken de ontwikkelstraat ook aan de hand van dit project. Dit gaan wij realiseren door alleen op de deelvragen in te gaan en door te experimenteren met de ontwikkelstraat gerelateerd aan de opdracht die wij van de derde partij krijgen.

Verantwoording

De reden voor het onderzoek

De reden van het onderzoek is dat het projectteam onvoldoende informatie heeft over het maken van een ontwikkelstraat. Daarom hebben wij een onderzoek opgezet om verschillende ontwikkelstraat technieken en technologieën met elkaar te vergelijken en hierover meer kennis op te doen.

Nieuwswaarde

Dit onderzoek is direct van waarde voor alle volgende IPSENH studenten. Deze studenten kunnen informatie putten uit ons onderzoek en deze lessen gebruiken voor haar eigen onderzoek. Daarnaast kan ons onderzoek ook worden gebruikt om bestaande ontwikkelstraat pakketten of combinaties op de proef te stellen en daarna eventueel te verbeteren.

Theoretische relevantie

Dit onderzoek geeft de studenten groepen van IPSENH veel kennis over het gebruik van ontwikkelstraten. Deze kennis is zeer nuttig bij het maken van kwalitatief hoogstaande software.

Praktische relevantie

De resultaten van dit onderzoek kunnen direct toegepast worden in het IPSENH-project. De inzichten die opgedaan worden in dit onderzoek geven de studenten een sterk inzicht in het gebruik van een ontwikkelstraat en welke voordelen dit met zich mee brengt. Ook voor later in de beroepspraktijk kan dit onderzoek helpen bij het verhogen van de kwaliteit van de code.

Maatschappelijke relevantie

De resultaten van dit onderzoek kunnen worden gebruikt door andere studenten of programmeurs die nog geen ervaring of kennis hebben van ontwikkelstraten in combinatie met codekwaliteit.

Reikwijdte en informatiegehalte

Kan de centrale onderzoeksvraag beantwoord worden?

De centrale onderzoeksvraag van dit onderzoek kan met een hard en concreet antwoord beantwoord worden. Codekwaliteit kun je op verschillende manier meten, het is daarbij alleen belangrijk dat er duidelijke criteria gesteld worden waaraan de verschillende resultaten aan moeten voldoen. Daarbij helpen analyses die gedaan worden door het projectteam, zoals de snelheid waarmee nieuwe features worden opgeleverd of hoelang het duurt voordat een bug geïdentificeerd is en opgelost.

Doelgroep

De directe doelgroep van dit onderzoek zijn de studenten van het project-IPSENH. Deze studenten gebruiken de resultaten van dit onderzoek voor het opzetten van een ontwikkelstraat. Daarnaast is de indirecte doelgroep: andere programmeurs die nog niet eerder met ontwikkelstraten gewerkt hebben om codekwaliteit te verbeteren.

Achtergrond

Achtergrond theorieën tot probleemstelling

Onderstaande theorieën hebben betrekking op de probleemstelling.

- Softwareontwikkeling is alle processen die samen een softwareproduct ontwikkelen. Deze processen kunnen zijn analyseren, specificatie requirements, programmeren, testen, documenteren en onderhouden. Softwareontwikkeling is niet alleen maar programmeren, voorbereiden van al je requirements is een heel belangrijk onderdeel van softwareontwikkeling.
- Code dat blijft doorontwikkeld en gebruikt wordt gezien als code van hoge kwaliteit, dus een code dat op lange termijn te handhaven is.

Kennisgebieden

Het kennisgebied waarin dit onderzoek zich bevindt is informatiekunde. Om specifieker te zijn, softwareontwikkeling. De kennisgebieden kunnen verspreid worden onder andere ontwerpen, programmeren, databases en software ontwikkelmethodes.

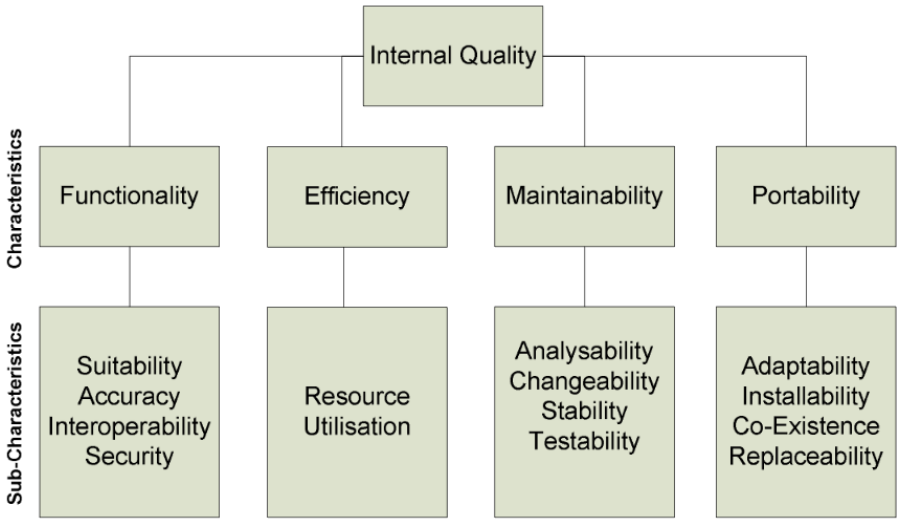
Inperking kennisgebied

Het onderzoek wordt gedaan op ontwikkelstraat en kwaliteit van code, het onderzoek blijft binnen het gebied van softwareontwikkeling en gaat niet verder dan dit.

Belangrijkste termen en concepten

Wat is volgens de literatuur codekwaliteit?

Volgens een artikel uit de International Journal of Software Engineering & Applications (Yiannis Kanellopoulos, 2010)



Figuur 1 Codekwaliteit Diagram

Codekwaliteit wordt vaak door een aantal onderdelen van de software bepaald. Dit zijn de karakteristieken die te zien zijn in de afbeelding. Hierbij staat “Internal Quality” gelijk aan code kwaliteit. Codekwaliteit houdt in dat de functionaliteit goed is, dat de code efficiënt is, dat de code onderhoudbaar is, en dat de code makkelijk overdraagbaar is. Al deze karakteristieken hebben ook weer een aantal onderliggende karakteristieken.

Om functionaliteit te definiëren halen we de volgende karaktereigenschappen erbij: geschiktheid, accuraatheid, onderlinge verwerkingswijze en beveiliging.
Om efficiëntie te definiëren halen we de volgende karaktereigenschap erbij: gebruik van hulpbronnen.
Om onderhoud baarheid te definiëren halen we de volgende karaktereigenschappen erbij: analyseerbaarheid, veranderbaarheid, stabiliteit en testbaarheid.
Als laatste om de overdraagbaarheid te definiëren halen we de volgende karaktereigenschappen erbij: Aanpasbaarheid, installeerbaarheid, de mogelijkheid om naast elkaar te bestaan en vervangbaarheid.

Wat er nog ontbreekt in het diagram zijn de volgende twee eigenschappen:
Gebruikersvriendelijkheid, het gaat hier met name om hoe makkelijk de software te begrijpen is.
Betrouwbaarheid, hierbij gaat het om de stabiliteit en om performance van de software. Het kan allemaal wel erg mooie code zijn, maar zonder deze twee eigenschappen is het niet bruikbaar.

Hier kunnen we nog een aantal zaken aan toevoegen volgens Peter J. Denning (Denning, 1992)

Volgens hem is het namelijk uitermate belangrijk dat de code dynamisch is, doordat het modulair is opgebouwd. Maar ook is het belangrijk dat er een uniforme stijl gebruikt wordt en dat er voldoende commentaar geschreven is.

Al deze karaktereigenschappen samengevoegd vormen een uitgebreide maar logische definitie voor codekwaliteit.

Hoe meet je code kwaliteit?

Object georiënteerd ontwikkelen is erg populair in de hedendaagse softwareontwikkeling. Het heeft aangetoond dat het waardevol is voor systemen die onderhouden en veranderd moeten worden. Objectgeoriënteerde software vergt een andere aanpak ten opzichte van functioneel programmeren wat betreft het evalueren van de kwaliteit van de software. De metingen van de kwaliteit van het traditionele functioneel programmeren wordt gemeten door te kijken naar de onafhankelijke datastructuur van de software. De kwaliteit van objectgeoriënteerde software richt zich op de combinatie van functies en data van geïntegreerde objecten. Of een metriek nieuw of oud is, is niet belangrijk. Het gaat er om dat het effectief moet zijn in het meten van de volgende attributen: efficiëntie, complexiteit, begrijpelijkheid, testbaarheid/onderhoud baarheid en hergebruik. Deze attributen evalueren de objectgeoriënteerde concepten: methods, classes, coupling en inheritance.

Er zijn twee soorten metrics: de traditionele en de objectgeoriënteerde metrics. In een object georiënteerd systeem worden de traditionele metrics meestal toegepast op de methoden. Onderstaand zijn de traditionele metrics:

- *Cyclomatic Complexity (CC)*
  Cyclomatic Complexity wordt gebruikt om de complexiteit van een algoritme of een methode te evalueren. Een methode met een lage cyclomatic complexity is over het algemeen beter. Een lage cyclomatic comlexity kan ook betekenen dat de methode operaties delegeert naar andere methodes, niet dat de methode niet complex is. Cyclomatic complexity kan niet worden gebruikt om de complexiteit van een klasse te meten. De complexiteit van individuele methoden kunnen worden gecombineerd worden met andere metrics om de complexiteit van een klasse te bepalen.
- *Size*
  De grootte van een methode wordt gebruikt om te evalueren of de methode makkelijk te begrijpen is voor ontwikkelaars. De grootte kan op verschillende manieren gemeten worden. Daarbij inbegrepen maar niet gelimiteerd tot het tellen van alle fysieke lijnen code, het aantal statements en het aantal lege regels. Hoewel de grootte verschilt per programmeertaal wordt het begrijpen van methoden moeilijker als deze groter is.

Veel verschillende metrics zijn voorgesteld voor het meten van objectgeoriënteerde software. De objectgeoriënteerde metrics die gekozen zijn door het SATC (Software Assurance Technology Center(NASA)) meten principes, die wanneer verkeerd ontworpen, een negatief effect hebben op de kwaliteit van de code.

- *Weighted Methods per Class (WMC).* De som van complexiteiten van methoden gedefinieerd in een klasse.
- *Response for a Class (RFC).* Het aantal methoden en constructors dat aangeroepen kan worden als er een bericht naar een object van die klasse gestuurd wordt.
- *Lack of Cohesion of Methods (LCOM).* Het geeft aan in hoeverre het single responsibility principle (SRP) wordt gehandhaafd binnen de software.
- *Coupling Between Object Classes (CBO).* Meet hoeveel klassen er met een specifieke klasse gekoppeld zijn. Het gaat hier om de methoden die de methoden van die klasse aanroepen.
- *Depth of Inheritance Tree (DIT).* Meet de positie van een klasse in de inheritance hiërarchie. Te veel inheritance kan voor problemen zorgen tijdens het ontwikkelen van software.

Al deze metrics gecombineerd geven een goed beeld van de kwaliteit van software.

Wat is een ontwikkelstraat?

Een ontwikkelstraat kan kort gedefinieerd worden als de structurele combinatie van tools, processen, componenten en methodes die de gehele softwareontwikkeling proces ondersteunen. Dus je kan zeggen dat een ontwikkelstraat een software development tool is of een omgeving ingericht voor software development.

Uit welke onderdelen bestaat een ontwikkelstraat?

Volgens een artikel uit (Santos, 2006) worden de belangrijkste onderdelen van een ontwikkelstraat onderscheiden als volgt:

- **Productlijn**
  Let op, dit is niet de productielijn, maar het type product. De productlijn wordt gedefinieerd als de verschillende varianten van producten dat een ontwikkelstraat kan produceren.

- **Schema**
  Het schema is de blauwdruk dat beschrijft wat de ontwikkelstraat zal ontwikkelen, de processtappen, welke producten, de configuraties ervan, hoe deze ontwikkeld moeten worden en de tools die nodig zijn voor de ontwikkelstraat.

- **Template**
  De template kan gezien worden als de implementatie van het schema. Deze implementatie bevat bijvoorbeeld de tools die je gaat gebruiken, code, frameworks, documentatie en andere middelen die gebruikt worden om een product te bouwen. Dit kan in de vorm van een installeerbaar pakket zijn. De template zorgt voor een startpunt voor de ontwikkeling van het project.

- **Product**
  Een product hier is wat de ontwikkelstraat produceert, het is ook belangrijk om te begrijpen dat dit geen compleet eindproduct hoeft te zijn zoals bijvoorbeeld Microsoft Word. Deze product kan later deel zijn van een groter product dat ontwikkeld wordt door een verschillend ontwikkelstraat.

Hoe kan je een ontwikkelstraat inrichten?

Voordat je een ontwikkelstraat gaat bouwen moet het duidelijk zijn waarom je hierin wilt investeren. Het inrichten van een ontwikkelstraat gaat eigenlijk over de invulling van het schema en de template. Wanner je een schema hebt beschreven en daarvan een template kunnen samenstellen heb je je ontwikkelstraat ingericht. Dit houdt in dat je een omgeving hebt ingericht dat gereed is om software te ontwikkelen met alle middelen ingesteld die hiervoor essentieel zijn.

Hoe kan je een ontwikkelstraat gebruiken om de codekwaliteit te verhogen?

Volgens Microsoft (Microsoft, 2016) heeft een ontwikkelstraat meerdere voordelen ten opzichte van code kwaliteit.

Ten eerste gaat het opzetten van een project gestroomlijnd door gebruik te maken van bewezen technieken. Dit heeft als effect op de code kwaliteit dat er minder fouten in de software en het ontwikkelproces kunnen komen dan wanneer er gebruik gemaakt wordt van niet bewezen technieken.

Ten tweede zorgt het ervoor dat de code getest kan worden door middel van automatische test-runs en regressie testen, zodat tijdens het hele proces de kwaliteit in termen van functionaliteit en code gewaarborgd kan worden.

Daarnaast kan de ontwikkelstraat volgens Octo (Octo, 2014) ook gebruikt worden om zelf regels op te stellen die bedrijfsspecifieke kwaliteitseisen en code regels afdwingen.

Ook kunnen de code architectuur regels die aan het begin zijn vastgesteld om de code leesbaarheid te waarborgen worden vastgesteld.

Door het continue proces van development, controle en opleveren van kleine onderdelen, worden nieuwe bugs en overtredingen van de regels snel opgemerkt. Dit zorgt ervoor dat de kwaliteit op geen enkel moment af.

Domeinen

Softwareontwikkeling

Onderzoek structuur en methode

Om de onderzoeksvraag en de deelvragen te beantwoorden gaan wij gebruik maken van onderzoeksmethode. De onderzoeksmethode die wij gaan gebruiken is “Desk Research”. We gaan literatuur van professionals gebruiken om onze vragen te beantwoorden. We gaan ook interviews afleggen om van mensen uit het beroepenveld kennis en meningen te kunnen evalueren en gebruiken voor ons onderzoek. Daarnaast gaan wij zelf experimenteren door stapsgewijs uit te voeren wat wij willen onderzoeken en hiervan de resultaten waarnemen en noteren. Hierbij gaan wij op verschillende aspecten meten, om zo concrete resultaten te creëren die kunnen worden gebruikt als onderbouwing van onze conclusie en advies. Dit zal onder andere: performance, en code coverage zijn.

Planning

We hebben 7 weken de tijd om het onderzoeksplan en het onderzoeksrapport te schrijven. We gebruiken de eerste twee weken om het onderzoeksplan op te stellen, en beginnen hierna aan het onderzoek. Na 3 weken onderzoeken, in week 5, beginnen we aan het verwerken van de resultaten in het onderzoeksrapport. Tussendoor worden de resultaten natuurlijk ook bijgehouden, maar deze resultaten kunnen nog worden verworpen. In week 6 beginnen we aan het adviesrapport, gebaseerd op het onderzoeksrapport.

Schematisch weergeven ziet dat er als volgt uit:

				Week					
	1	2	3	4	5	6	7		
Onderzoeksplan									
Onderzoek									
Onderzoeksrapport									
Adviesrapport									

Bibliografie

Denning, P. J. (1992). What is Software Engineering? *A Commentary from Communications of ACM*.

Greenfield, J. (2004, November). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Retrieved from Microsoft: https://msdn.microsoft.com/en-us/library/ms954811.aspx

Husu, M. (2006). Software Factories. *Seminar on Service-Oriented Software Engineering*, 1-19. Retrieved from CiteSeerX: http://citeseerx.ist.psu.edu/viewdoc/similar?doi=10.1.1.131.7889&type=ab

Lenz, G. (2008). *An Introduction to Software Factories*. Retrieved from Dzone: https://dzone.com/articles/an-introduction-software-facto

Microsoft. (2016, 08 26). *Software Factories*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/ff699235.aspx

Octo. (2014, 06 30). *Toward a better software factory*. Retrieved from Octo Talks: http://blog.octo.com/en/toward-a-better-software-factory/

Santos, J. (2006, November 3). *What is a Software Factory?* Retrieved from Micrososft: https://blogs.msdn.microsoft.com/jezzsa/2006/11/03/faq-what-is-a-software-factory/

Yiannis Kanellopoulos, P. A. (2010). Code Quality Evaluation Methodology Using The ISO/ IEC 9126 Standard. *International Journal of Software Engineering & Applications Nr 3*, 1-20.