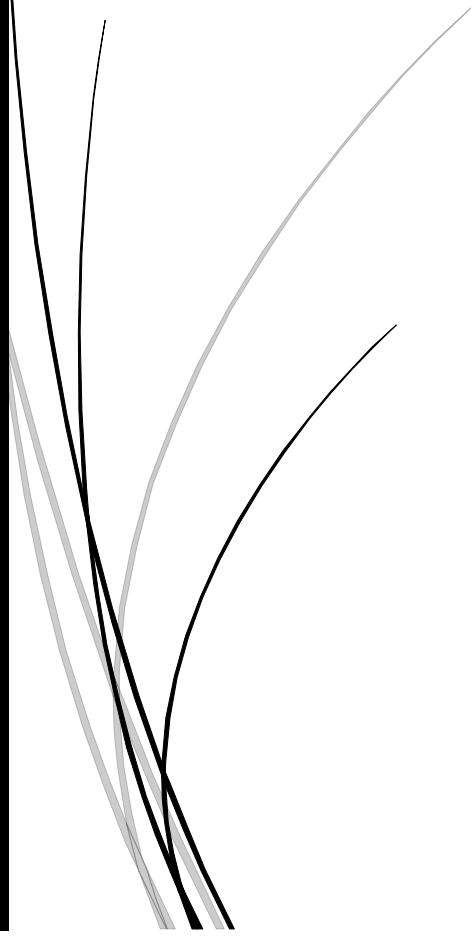
26-10-2016

Onderzoeksrapport

IPSENH



RZN

REPOSITORY ZONDER NAAM

Versiebeheer

Versie	Datum	Omschrijving	Auteur
0.1	20-09-2016	Initiële versie	Anton
0.2	02-10-2016	Eerste invulling Opzet & resultaten	Anton
0.3	03-10-2016	Aanpassing resultaten codekwaliteit, beperkte invulling andere hoofdstukken	Anton
0.4	04-10-2016	Correcties en invulling hypothese probleemstelling	Anton
0.5	04-10-2016	Opzet onderzoek toegevoegd.	Sander
0.6	04-10-2016	Onderzoek ontwikkelstraat	Zairon
0.7	05-10-2016	Controle hoofdstuk Zairon en feedback	Anton, Sander
0.8	07-10-2016	Onderzoek ontwikkelstraat onderdelen aangepast	Zairon
0.9	09-10-2016	Onderzoek ontwikkelstraat onderdelen bijgewerkt	Zairon
0.10	11-10-2016	Invulling suggesties verder onderzoek	Anton
0.11	11-10-2016	Invulling opzet onderzoek	Sander
0.12	11-10-2016	Invulling meet attributen opzet experiment	Zairon, Anton
0.13	11-10-2016	Bronnen toegevoegd	Sander
0.14	12-10-2016	Onderzoek IDE’s	Anton
0.15	12-10-2016	Onderzoek Code Analysis	Anton, Sander
0.16	12-10-2016	Onderzoek Build Automation	Anton, Laurens
0.17	12-10-2016	Onderzoek Version Control	Laurens
0.18	13-10-2016	Reactie Peter verwerken	Anton
0.19	25-10-2016	Tools onderzoek ingevuld voor angular	Zairon, Laurens
0.20	25-10-2016	Aanpassing hoofdstukken 3,4, 8, 10	Anton
0.21	26-10-2016	Opzet aanbevelingen	Laurens
0.22	26-10-2016	Invulling samenvatting	Anton
1.0	26-10-2016	Afronding voor oplevering	Anton, Zairon, Laurens, Sander

1 Inhoud

2 Samenvatting

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

Deze vraag hebben wij onderzocht tijdens dit onderzoek. Om deze vraag te beantwoorden hebben wij een aantal deelvragen opgesteld. Deze deelvragen hebben we voornamelijk met literatuuronderzoek beantwoord. Naast deze onderzoeksvraag hebben wij ook in verband met een opdracht van 42Windmills een onderzoek uitgevoerd naar de tools die wij zouden kunnen gebruiken om de ontwikkelstraat op te zetten. Hier komt een aanbeveling uit voort. De deelvragen zijn hieronder kort beantwoord.

Wat verstaat men onder codekwaliteit?
Codekwaliteit is erg belangrijk. De belangrijkste punten van codekwaliteit zijn vooral dat het leesbaar en onderhoudbaar is met als doel om makkelijk overdraagbaar te zijn tussen programmeurs en teams.

Hoe meet je codekwaliteit?
Codekwaliteit meten wordt gedaan door te kijken naar de complexiteit, de grootte, de moeilijkheidsgraad van relaties en hoeveel functies een klasse heeft. Dit wordt meestal met tools gedaan.

Uit welke onderdelen bestaat een ontwikkelstraat?
De ontwikkelstraat bestaat uit *Integration Tool, Build Automation, Automated Tests, Version Control, Code Review, Code Analysis, IDE en optioneel nog monitoring tools.*

Hoe kan je een ontwikkelstraat inrichten?
Om een ontwikkelstraat in te richten maak je gebruik van de bovenstaande onderdelen, hier gebruik je per onderdeel een tool voor die aansluit op de andere onderdelen. Deze richt je in voor de situatie waar hij voor nodig is.

Hoe kan je een ontwikkelstraat gebruiken om de codekwaliteit te verhogen?
Wanneer er consequent gebruik wordt gemaakt van alle onderdelen uit de ontwikkelstraat resulteert dit bijna vanzelf in een hogere codekwaliteit. De verschillende onderdelen voegen ieder een stuk zekerheid aan de code toe, doordat fouten vroegtijdig kunnen worden gevonden en opgelost.

We raden aan om voor C# en Angular2 / JavaScript een ontwikkelstraat op te stellen bestaande uit de volgende opstelling:

Integration Tool: Jenkins
Build Automation: MSBuild (C#), Grunt (Angular2, JS)
Automated Tests: NUnit (C#), Mocha (Angular2, JS), Protractor (Angular2, JS)
Version Control: Git
Code Review: GitHub
Code Analysis: SonarQube (C#), JSLint (JS)
IDE: Vistual Studio (C#), WebStorm (Angular2 / JS)

3 Inleiding

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 o.a. 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.

Wat wij willen bereiken met het onderzoek is het duidelijk 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.

Wij doen dit onderzoek voor 42Windmills en onszelf, maar andere studenten die nog geen ervaringen hebben met ontwikkelstraten kunnen ook goed gebruik maken van de resultaten. De doelgroep is dus voornamelijk programmeurs die meer willen weten van ontwikkelstraten en codekwaliteit.

We voeren dit onderzoek uit voordat wij een opdracht voor 42Windmills gaan uitvoeren. Zij maken voornamelijk gebruik van Microsoft tools, maar zij willen ook Angular2 gaan gebruiken. Daarom doen wij dit onderzoek voornamelijk gerelateerd aan de taal C# en Angular2.

Wij geven in het hoofdstuk probleemstelling aan wat precies onze onderzoeksvragen zijn, en hoe we dit willen beantwoorden door middel van deelvragen.

Het document is opgebouwd in een aantal hoofdstukken. We omschrijven eerst een kort beeld van de literatuur in de omgeving, specificeren daarna de probleemstelling, omschrijven onze hypothese. Daarna omschrijven we de gevonden resultaten en op basis hiervan trekken wij een conclusie en doen wij aanbevelingen.

4 Literatuurverkenning

Er is voor de verschillende deelvragen literatuur te vinden per onderwerp. Deze literatuur is vrij oud en de literatuur plaatst zich niet in het concept van ontwikkelstraten. Wat wij met dit onderzoek doen is codekwaliteit en de ontwikkelstraat samenvoegen, om zo een witte vlek in de literatuur te vullen. Per onderwerp is er al een hele hoop literatuur geschreven, omdat een groot deel van de softwareontwikkelaars deze concepten erg belangrijk vinden, en ook gebruik maken hiervan. We zoeken literatuur van professionele schrijvers, maar we gebruiken ook artikelen geschreven door ervaren mensen om zo het aanbod te vergroten.

5 Probleemstelling en begrippenlijst

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

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

We gaan deze onderzoeksvraag beantwoorden door de volgende deelvragen te beantwoorden:

- *Wat verstaat men onder codekwaliteit?*
- *Hoe meet je codekwaliteit?*
- *Uit welke onderdelen bestaat een ontwikkelstraat?*
- *Hoe kan je een ontwikkelstraat inrichten?*
- *Hoe kan je een ontwikkelstraat gebruiken om de codekwaliteit te verhogen?*

5.1 Begrippenlijst

JS : JavaScript
Angular2: Framework gemaakt door google op de taal JavaScript
IDE: Integrated Development Environment
Repository: Plaats waar broncode kan worden opgeslagen
GUI: Graphical User Interface, wat de eindgebruiker ziet en waar acties op worden uitgevoerd.

6 Theorie en hypothese

Onze hypothese is: “Wij verwachten dat een ontwikkelstraat met onderdelen specifiek gekozen voor de toepassing ervoor zorgt dat de codekwaliteit hoger is, door alle hulpmiddelen die de ontwikkelstraat biedt, en door de werkwijze die gehanteerd wordt”.

7 Opzet en uitvoering van het onderzoek

Om het onderzoek uit te zoeken hebben wij per onderzoeksvraag een aanpak bedacht. We gebruiken met name literatuuronderzoek. Voor de onderzoeksvraag: “Wat is codekwaliteit?” voeren we ook interviews uit met een aantal experts, in dit geval zijn dit er drie. De gestelde vragen zijn te vinden in bijlage A.

Het onderzoek typeert zich voornamelijk als een verklarend onderzoek. Wij zijn erop uit om te achterhalen hoe iets bijdraagt. Om dit te onderzoeken maken wij gebruik van digitale bronnen, deze verkrijgen wij via verschillende zoekmachines zoals Google Scholar, maar ook databanken. Deze bronnen gebruiken wij vervolgens in de resultaten van de deelvragen en hier verwijzen wij naar per deelvraag.

De resultaten van de deelvraag over de onderdelen van de ontwikkelstraat gebruiken wij om per onderdeel een lijst van tools te zoeken. Met deze tools zullen we een matrix opstellen die overzichtelijk laat zien welke tools wij mogelijk kunnen gebruiken per onderdeel van de ontwikkelstraat.

We doen onderzoek naar C# en Angular2 omdat wij na dit onderzoek gebruik moeten maken van deze talen om een opdracht te vervullen. Wij willen graag de resultaten van dit onderzoek kunnen gebruiken hiervoor. Daarnaast kun je tools voor verschillende talen niet met elkaar vergelijken dus moest er sowieso een beperking op talen worden gelegd.

Hieronder per deelvraag de opzet:

7.1 Opzet interview codekwaliteit en ontwikkelstraat

Er wordt van tevoren een interview geschreven die wordt afgelegd met personen uit het werkveld. De resultaten hiervan worden gebruikt als bron om antwoord te geven op de deelvragen. De vragen en de exacte resultaten zijn opgenomen als bijlage.

7.2 Opzet literatuuronderzoek meten codekwaliteit

Voor dit deel van het onderzoek gaan we vooral op zoek naar onderzoeken die al uitgevoerd zijn op dit gebied. Ook de meningen van invloedrijke personen (Kent Beck, Robert C. Martin) worden meegenomen in het literatuuronderzoek om uiteindelijk te bepalen wat belangrijk is bij het meten van de kwaliteit van code.

7.3 Opzet literatuuronderzoek onderdelen van een ontwikkelstraat

Voor dit gedeelte van het onderzoek wordt er gebruik gemaakt van verschillende bronnen die te vinden zijn op het internet over ontwikkelstraat, Continuous Integration, Continuous Delivery en Continuous Deployment. Deze kunnen artikels zijn, boeken, een website of wetenschappelijke bronnen. Met deze kennis zullen we antwoord geven op de deelvraag.

7.4 Opzet experiment tools per onderdeel van de ontwikkelstraat

Om uit te zoeken welke tool het beste scoort per onderdeel van de ontwikkelstraat moeten wij eerst kijken naar hoe wij deze tools kunnen vergelijken met elkaar. Op welke aspecten we deze tools beoordelen, en hoe we hier een concrete waarde aan hangen. Om dit te testen gaan we met test code de verschillende tools testen op de vooraf gestelde eisen. Deze eisen zijn nog niet bepaald en worden na het beantwoorden van de deelvragen duidelijk.

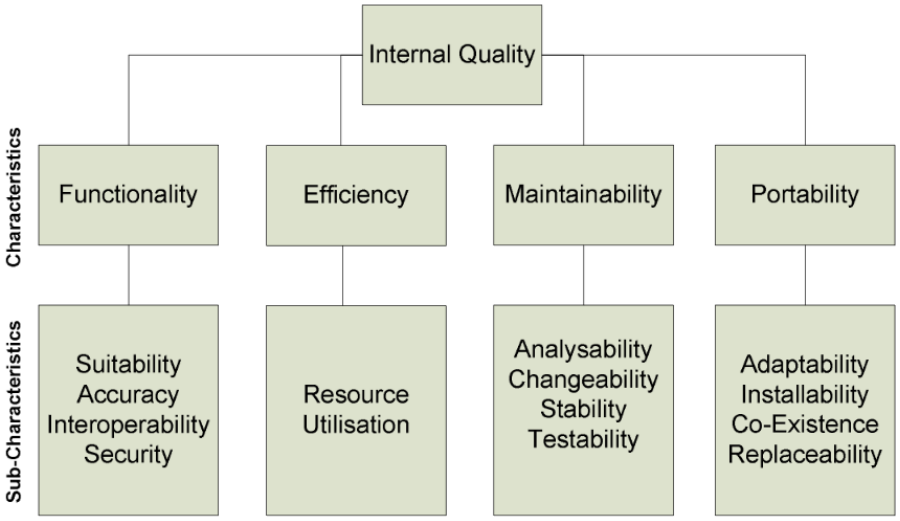
8 Resultaten

Hier wordt per deelvraag een antwoord gegeven. Deze antwoorden zullen wij gebruiken om de uiteindelijke conclusie op te baseren, en om grond te geven voor het advies.

8.1 Wat is codekwaliteit?

Het doel van codekwaliteit is gebruiksgemak, gebruiksgemak voor de ontwikkelaar. Als de ontwikkelaars makkelijk kunnen werken met een grote codebase, hierdoor sneller aan de slag kunnen met de code geschreven door anderen, wordt de productiviteit hoger.

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



Figuur 1 Codekwaliteit Diagram

Codekwaliteit wordt vaak door een aantal aspecten van de code 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.

Experts nemen codekwaliteit erg serieus, en vinden codekwaliteit erg belangrijk, in sommige gevallen zelfs zo belangrijk dat er een stukje performance (grootte van variabele namen etc.) kan worden opgeofferd om de codekwaliteit te verhogen.

Bedrijven streven naar een zo hoog mogelijke codekwaliteit omdat zij op deze manier makkelijker nieuwe programmeurs aan hun software kunnen laten bouwen, en toekomstige wijzigingen aankunnen.

Code hoort elegant en efficiënt te zijn. De logica moet duidelijk zijn, alsof er geen andere mogelijkheid te bedenken is. De code moet testbaar zijn, leesbaar maar ook opgebouwd zijn uit kleine deelsystemen. De code moet niet te complex zijn, hiermee wordt bedoeld dat de code niet moet laten zien hoe slim jij bent, maar hoe goed jij bent in andere mensen jouw code te laten begrijpen.

De code hoort opgebouwd te zijn uit korte methodes, korte classes en zonder duplicaten waardoor de code makkelijker is te testen en te lezen. Een ander aspect van leesbaarheid is het gebruik van juiste variabel namen en uitgebreid commentaar waar nodig. De code leest ook makkelijker wanneer iedereen zich aan dezelfde stijl-standaarden houdt (spaties versus tabs, accolades op dezelfde regel versus op een nieuwe regel, etc.).

Code kwaliteit kun je realiseren door gebruik te maken van hulpmiddelen, zowel automatisch als handmatig. Er kunnen syntax en formatteer checks worden gedraaid op alle code, dit zorgt ervoor dat er correcte code is, en dat de stijl uniform is. Daarnaast kun je gezamenlijke code-reviews houden om zo met elkaar stukken code te verbeteren. Als laatste kunnen er tests worden geschreven die ervoor zorgen dat alle code die gemaakt wordt ook daadwerkelijk werkt. Iedere keer wanneer er wijzigingen gemaakt worden kunnen deze tests worden gedraaid, als controle.

Bronnen: Peter J. Denning (Denning, 1992), Clean Code (Martin, 2008), Interviews Aiko Mastboom, Kah Loon Yap, Peter van Vliet (zie bijlage)

8.2 Hoe meet je code kwaliteit?

Objectgeoriënteerde software vergt een andere aanpak ten opzichte van functioneel programmeren wat betreft het evalueren van de kwaliteit van de software. Het meten van de kwaliteit van het traditionele functioneel programmeren wordt gerealiseerd door te kijken naar de onafhankelijke data- of ontwerpstructuur van de software. De kwaliteit van objectgeoriënteerde software richt zich op de combinatie van functies en data van geïntegreerde objecten. De resultaten van de metingen worden in de vorm van software metrics gepresenteerd. Of een metric nieuw of oud is, is niet belangrijk. Het gaat er om dat het effectief moet zijn in het meten van één of meer van de volgende attributen: functionaliteit, betrouwbaarheid, bruikbaarheid, efficiëntie, onderhoudbaarheid en overdraagbaarheid (ISO/IEC 25010:2011 - Systems and software engineering, 2011). Deze attributen evalueren de objectgeoriënteerde concepten: methods, classes, coupling en inheritance.

Er zijn twee soorten metrics: de traditionele en de objectgeoriënteerde metrics. In een objectgeoriënteerd systeem worden de traditionele metrics meestal toegepast op de methoden. De onderstaande zijn de traditionele metrics.

- **Cyclomatic Complexity(CC)** wordt gebruikt om de complexiteit van een algoritme in een methode te evalueren. Een methode met een lage cyclomatic complexity is over het algemeen beter. Een lage cyclomatic complexity 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, maar de complexiteit van individuele methoden kan worden gecombineerd met andere metrics om de complexiteit van een klasse te bepalen.
- **Size** evalueert of de methode makkelijk te begrijpen is voor ontwikkelaars. De grootte kan op verschillende manieren gemeten worden. Daarbij zit inbegrepen maar is 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, het een negatief effect heeft op de kwaliteit van de code. Twee studenten van Växjö Universiteit te Zweden hebben de meest voorkomende class (OO) gerelateerde metrics die tools gebruiken onderzocht (Rüdger, Jonas, & Welf, 2008). De metrics die uit het genoemde onderzoek naar voren zijn gekomen worden hieronder verder toegelicht.

- **Coupling Between Object classes (CBO)** is een telling van het aantal andere klassen waaraan een specifieke klasse is gekoppeld. Het is gebaseerd op het aantal niet-overervende gerelateerde klasse hiërarchieën waarvan een klasse afhankelijk is. Het overmatig gebruik van coupling schadelijk aan het modulaire design en gaat hergebruik tegen. Hoe onafhankelijker een klasse is, hoe makkelijker het in een andere applicatie gebruikt kan worden. Een hoog aantal koppelingen zorgt ervoor dat de software gevoelig wordt voor veranderingen in andere delen van de software. Onderhoud is hierdoor ook moeilijker. Daarnaast zorgt veel coupling voor een ingewikkeld systeem die moeilijk te begrijpen is. Het doel is om zo zwak mogelijk te koppelen tussen modules. CBO evalueert efficiëntie en hergebruik.
- **Depth of Inheritance Tree (DIT)** is de diepte van een klasse in de inheritance hiërarchie is de maximale lengte van een klasse tot de root van de hiërarchie. Het wordt gemeten door het aantal voorgaande klassen te tellen. Hoe dieper een klasse zich in de hiërarchie bevindt, hoe meer kans er is dat de methoden in de klasse overerven van een klasse bovenin de hiërarchie. Dit maakt het complexer om het gedrag van een klasse te bepalen. Grote hiërarchieën brengen meer complexiteit met zich mee, omdat er meer klassen en methoden bij betrokken zijn. Deze metric evalueert de efficiëntie en hergebruik, maar ook testbaarheid en begrijpbaarheid.
- **Lack of Cohesion of Methods (LCOM)** meet in hoeverre methoden in klassen met elkaar samenwerken om goede resultaten te produceren. Een effectief objectgeoriënteerd design maximaliseert de samenwerking tussen methoden, omdat het encapsulatie stimuleert. Een hoge samenhang betekent dat klassen goed onderverdeeld zijn. Klassen met lage samenhang kunnen mogelijk in meerdere subklassen gesplitst worden die zelf een hogere samenhang hebben. LCOM meet efficiëntie en hergebruik.
- **Number of Children (NOC)** is het nummer wat telt hoeveel subklassen een klasse onder zich heeft in de klassenhiërarchie. Het is een mogelijke indicatie van de invloed die een klasse heeft op het design en de software. Hoe groter het aantal subklassen, des te meer kans op slechte abstractie van de parent klasse. Dit kan betekenen dat er slechte subklassen gemaakt zijn, die misschien niet nodig geweest waren. Echter, hoe meer subklassen, hoe groter het hergebruik van deze klassen zal zijn. Als een klasse een groot aantal subklassen heeft, kan dit betekenen dat er meer getest moet worden. Hierbij wordt de testtijd dus vergroot. Deze metric richt zich vooral op efficiëntie, hergebruik en testbaarheid.
- **Response For a Class (RFC)** is het aantal methoden die aangeroepen kunnen worden als gevolg van een call op een object van de desbetreffende klasse of methode van dit object. Dit omvat alle methoden die beschikbaar zijn in de klassen hiërarchie. Deze metric kijkt naar een combinatie van complexiteit van een klasse door het aantal methoden en het aantal communicaties met andere klassen te vergelijken. Hoe groter het aantal methoden dat aangeroepen kan worden is, hoe complexer de klasse is. Als er een groot aantal andere klassen en methoden aangesproken kunnen worden, dan maakt dit het lastig voor de tester, omdat het debuggen veel complexer wordt. Deze metric evalueert begrijpbaarheid, onderhoudbaarheid en testbaarheid.

- **Weighted Methods per Class (WMC)** is het aantal geïmplementeerde methoden in een klasse of de som van de complexiteiten van alle methoden in een klasse (met behulp van cyclomatic complexity). Dit tweede is lastig te implementeren, omdat niet alle methoden in een klasse benaderbaar zijn vanwege overerving. Hoe groter het aantal methoden in een klasse, hoe groter de impact op de subklassen kan zijn, omdat alle subklassen de methoden overerven. Klassen met veel methoden zijn vaak applicatie specifiek. Dat wil zeggen dat het niet goed te hergebruiken is. Deze metric evalueert de begrijpbaarheid, onderhoudbaarheid en hergebruik.

8.3 Wat is een ontwikkelstraat?

Een softwareproject moet op tijd klaar zijn en binnen budget blijven om succesvol te zijn. In werkelijkheid is dit niet altijd het geval, de meeste softwareprojecten falen. Het kan zijn dat het eindproduct helemaal niet overeenkomt met de verwachtingen. Vroeger werden auto's met de hand gebouwd en dit nam veel tijd in beslag. Toen heeft Henry Ford de "assembly lines" ontworpen, dit is een wijze om auto's geautomatiseerd te bouwen. De machines die hij heeft ontwikkeld hiervoor hebben een massaproductie van auto's kunnen bereiken in een korte tijd en met lagere kosten. Waarom deze concepten niet omzetten naar software development? Elke softwareproject is verschillend, maar ze bevatten allemaal onderdelen die gemeenschappelijk zijn. Denk hier aan webpagina's, services, data access, securitymanagement, logging etc. Dus het idee hier is om een Software Factory te bouwen (Fernandes, 2013). In het Nederlands is dit vertaald naar ontwikkelstraat.

Volgens Jack Greenfield wordt een ontwikkelstraat gedefinieerd als volgt: *"A Software Factory is a softwareproduct line that configures extensible tools, processes and content using a software factory template based on a software factory schema to automate the development and maintenance of variants of an archetypical product by adapting, assembling and configuring framework-based components* (Jack Greenfield, 2004)*."*

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. Een belangrijke reden om deze omgeving in te richten is om het gehele proces te automatiseren en zorgen dat producten op tijd af zijn.

Continuous Integration, Continuous Delivery en Continuous Deployment kunnen je helpen om de processen en tools die je nodig hebt te verzamelen voor het automatiseren van de ontwikkeling.

8.3.1 Continuous Integration

Continuous Integration betekent dat je jouw code vaker gaat integreren, meestal dagelijks. Bij ieder integratie wordt de code gecontroleerd door een automation build, er worden ook tests gedaan op de code. Dus bij Continuous Integration wordt je code continu gecontroleerd tijdens het ontwikkelingsproces (Fowler, Continuous Integration, 2006).

8.3.2 Continuous Delivery

Bij Continuous Delivery ben je altijd in staat om een deployment te doen naar productie, maar je kan ervoor kiezen om dit niet te doen. Continuous Integration ging over integratie, testen en builden van code binnen development. Continuous Delivery bouwt zichzelf hierop en zorgt voor de laatste fasen van productie deployment (Fowler, ContinuousDelivery, 2013).

8.3.3 Continuous Deployment

Continuous Deployment kan gedefinieerd worden als veranderingen aan de software die door de Deployment Pipeline gaan en automatisch naar productie worden gezet. Het resultaat hiervan is dat je veel productie deployments krijgt elke dag. Om Continuous Deployment uit te voeren moet je ook bezig zijn met Continuous Delivery (Fowler, ContinuousDelivery, 2013).

8.4 Uit welke onderdelen bestaat een ontwikkelstraat?

Een ontwikkelstraat bestaat uit verschillende onderdelen die het software ontwikkelingsproces ondersteunen. Deze zijn tevens de onderdelen die Continuous Integration, Continuous Delivery en Continuous Deployment ondersteunen. Deze onderdelen zijn de tools die de software ontwikkelingsproces zal automatiseren, welke de exacte tools zijn die je gaat gebruiken hangt ook af van de type product. In een ontwikkelstraat wil je de ontwikkeling van software zo goed mogelijk automatiseren, hierdoor spelen de combinatie van tools die hieronder genoemd zijn een belangrijke rol in een ontwikkelstraat.

- Integration Tool
- Build Automation
- Automated Tests
- Version Control
- Code Review
- Code Analysis
- IDE
- Orchestration
- Dashboards
- Monitoring

Tools gehaald uit (Srinivasan, 2016) en (Continuous Delivery Tools List, 2011). Per onderdeel wordt nu kort toegelicht wat het inhoudt.

8.4.1 Integration Tool

Een integration tool wordt gebruikt om Continuous Integration te ondersteunen. Deze tool bewaakt de source code repository en bij het treffen van veranderingen wordt meteen een nieuwe build gestart. Tijdens deze build worden er tests gedaan op de code en als fouten ertussen zitten wordt er een notificatie gestuurd naar de betreffenden. Natuurlijk wordt de build proces hierdoor niet afgemaakt, bij het niet treffen van fouten krijg je een succesvolle build. Deze tool maakt het vinden en verwijderen van bugs makkelijk door de code zo vroeg mogelijk te testen, en hierdoor worden de risico’s ook vermindert (Srinivasan, 2016).

8.4.2 Build Automation

Een belangrijk onderdeel van Continuous Integration is Build Automation. Deze tool wordt gebruikt om de gehele build proces te automatiseren in plaats van deze handmatig te doen elke keer. Een build tool maakt een executable van de source code door deze te compileren, linken en verpakken. Een build tool is vooral geschikt bij grote projecten om bij te houden wat gebouwd moet worden, in welke volgorde en welke dependencies erbij horen (Build Tool, n.d.).

8.4.3 Automated Tests

Test Automation is ook een belangrijk onderdeel van Continuous Integration. Hiervoor zijn er verschillende tools die je kan gebruiken. Deze tool draait testen op een applicatie voordat deze naar productie gaat. Test Automation zorgt ervoor dat geen fouten zitten in nieuwe versies van applicaties, dit door bij elke verandering van de applicatie tests te laten draaien. De ontwikkelaar kan op dit manier snel zien als de functionaliteiten van de applicatie goed blijven werken na elke verandering. Vier soorten van testing tools die je kan gebruiken zijn automated web testing tools, automated gui testing tools, unit testing frameworks en automated testing cloud services (Software Test Automation Tools, n.d.).

8.4.4 Version Control

Version Control is een belangrijk onderdeel van Continuous Integration. Version Control wordt gebruikt om alle versies van de code en features bij te houden. Degenen die verantwoordelijk zijn voor veranderingen die gedaan worden wordt ook bijgehouden. Als er een fout ontdekt is kan men makkelijk terugdraaien naar een eerdere versie van de code om fouten te corrigeren, dit zonder anderen te storen. Teamleden kunnen tegelijkertijd werken op hun eigen branch en op een later moment deze branch laten vergelijken met andere veranderingen die gedaan zijn door anderen om te zorgen dat er geen conflicten ontstaan (Srinivasan, 2016).

8.4.5 Static Code Analysis

Static code analysis is het debuggen van de code zonder deze te uitvoeren. Deze tool zorgt dat de code voldoet aan de standaarden. Static Code Analysis kan fouten vinden die zichzelf niet vertonen tot een later moment. Dit soort van analysis behoort bij white box testing. Volgens experts is het compileren van code door compilers een manier van Static Code Analysis, omdat deze technische en syntactisch fouten kan vinden (Rouse, n.d.).

8.4.6 Code Review

Code review betekent dat je je code laat controleren door iemand anders. Hiervoor zijn er ook tools die je kan gebruiken om code te reviewen. Met een Code Review tool kan je zien welk stukje van de code veranderd is of aangeven welk stukje goed is of niet. Met deze tool kan je het stukje code dat veranderd is vergelijken met de huidige code base. Code review helpt om fouten vroeg te ontdekken en de kwaliteit van het product te houden (Srinivasan, 2016).

8.4.7 IDE

IDE staat voor Integrated Development Environment. Deze programmeeromgeving helpt de ontwikkelaar met het bouwen van applicaties en het maximaliseren van de productiviteit door enkele ingebouwde tools/functionaliteiten te bieden die het proces faciliteren. Sommige IDE's kunnen meer dan één programmeertaal ondersteunen. Een IDE bevat bijvoorbeeld een tekstverwerker, een compiler, een debugger, en komt met een GUI (Integrated Development Environment (IDE), n.d.).

8.4.8 Orchestration

Orchestration is het uitrollen van applicaties in de juiste volgorde in een geconfigureerde omgeving. Dit houdt in het beheren van veranderingen, updates of het afmaken van applicaties in de juiste volgorde. Door deze taak te automatiseren kan je menselijke fouten verminderen (Srinivasan, 2016).

8.4.9 Dashboards

Het doel van deze tool is om de status van de gehele omgeving bekend te maken voor andere ontwikkelaars of mensen die hieraan behoefte hebben. Bijvoorbeeld als een eerder gedaan test succesvol was of niet, wat actueel in productie zit etc. De status van elke machine/ server binnen de omgeving worden ook getoond in het dashboard. Iedereen binnen het team moet kunnen zien wat aan het gebeuren is. Bij sommige Integration Tools komt dit functionaliteit ingebouwd (Srinivasan, 2016).

8.4.10 Monitoring

Deze tool wordt gebruikt om verschillende soorten data te verkrijgen in een testomgeving of andere omgevingen en deze analyseren. Deze data kan gebruikt worden om te kunnen zien als er fouten ontstaan, als de prestatie van de applicatie aan het verbeteren is of niet, en wat hieraan gedaan kunt worden (Srinivasan, 2016).

8.5 Hoe kan je een ontwikkelstraat inrichten?

Volgens het boek (Weber, 1997, p. 40) moet een ontwikkelstraat ingericht worden volgens de specifieke requirements, er zal nooit een situatie zijn waar je een ontwikkelstraat zomaar kunt kopen van een leverancier en gebruiken zonder deze te modificeren. Weber vertelt dat het inrichten van een ontwikkelstraat 4 fasen heeft:

- Het voorafgaand begrijpen van de specifieke requirements;
- Eerste processen definiëren van de ontwikkelstraat;
- De juiste middelen kiezen om deze processen te ondersteunen;
- Trainen voor het gebruik van processen en middelen.

8.5.1 Het implementeren van Continuous Integration, Continuous Delivery en Continuous Deployment

Het beste is om Continuous Integration vroeg te implementeren, omdat het moeilijker is om dit te doen tijdens de latere fasen van een project. Als je laat begint met het implementeren van Continuous Integration is het aanbevolen om niet in één keer alles doen, maar begin met de kleine onderdelen. Er zijn vele manieren om Continuous Integration in te richten, het kan zijn dat je in het begin een shell script laat draaien en hierna overstapt naar een automation build tool. Het belangrijkste is dat bij elke verandering aan het systeem, een build gestart wordt. Je kan beginnen met een build laten draaien bij elke verandering aan het systeem, zonder tests te draaien. Als deze proces goed loopt en je een goede testing tool hebt gevonden kan je beginnen met het uitvoeren van tests bij de build proces. Continuous Integration gaat niet alleen over de technische implementatie, het gaat over het idee achter het concept (Duvall, 2007). Drie belangrijke dingen die je moet hebben om te beginnen met Continuous Integration is version control, automation build en een overeenkomst met het team, dit laatste omdat Continuous Integration geen tool is, maar een werkwijze. Dus je hebt per se geen Integration software nodig, maar het is wel handig (Farley & Humble, 2010).

Continuous Delivery is de vervolgstap van Continuous Integration, als je Continuous Integration hebt ingericht kan je beginnen met Continuous Delivery en ook Continuous Deployment. Continuous Delivery en Continuous Deployment zijn een uitbreiding op wat Continuous Integration doet.

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

Volgens Allan Kelly van DZone (Allan Kelly, 2010) kan een ontwikkelstraat op meerdere manieren worden ingezet om een bijdrage te geven aan het verbeteren van code kwaliteit.

8.6.1 Integration Tool

Een integration tool kan worden gebruikt om bij elke oplevering te controleren of de gehele codebase voldoet aan de architectuureisen en of er geen compilatie en vormgevings fouten inzitten. Dit zorgt ervoor dat de code kwaliteit niet ongemerkt af kan nemen. (Srinivasan, 2016).

8.6.2 Build Automation

Build automation zorgt ervoor dat er geen menselijke of volgorde fouten in het proces kunnen sluipen. Hierdoor wordt het opleverproces ook geautomatiseerd en alles wat niet aan de gestelde eisen van opleverkwaliteit voldoet kan niet worden opgeleverd. (Build Tool, n.d.).

8.6.3 Automated Tests

Test automation zorgt ervoor dat alle functionele eigenschappen van de software niet in waarde af kunnen nemen. Dit zorgt ervoor dat ongeteste code, welke bekend staat om fouten te introduceren die pas op een later moment ontdekt en opgelost worden, niet ongemerkt opgeleverd kan worden. (Software Test Automation Tools, n.d.).

8.6.4 Version Control

Version control zorgt ervoor dat er een tijdlijn opgebouwd wordt waarin de progressie van de codebase kan worden gemonitord en kan worden terug gezet wanneer deze afneemt. Ook kan gekeken worden wanneer eventuele fouten zich voor het eerst voor deden en hieruit kan lering getrokken worden hoe dit in de toekomst voorkomen kan worden. (Srinivasan, 2016).

8.6.5 Static Code Analysis

Static code analysis is een sterke tool voor het analyseren, meten en verbeteren van code kwaliteit. Dit kan ingezet worden door standaard regels en gebruiken aan te houden die code kwaliteit beoordelen. Aan de hand van deze beoordeling kan actie worden ondernomen om de code kwaliteit weer op het oude niveau te brengen. Wanneer dit succesvol is geimplementeerd in een ontwikkelstraat, kunnen de regels worden de regels constant bijgewerkt en aangevuld om ervoor te zorgen dat de code kwaliteit standaard verhoogd wordt en meegaat met zijn tijd. (Rouse, n.d.).

8.6.6 Code Review

Bij een Code review word indirect de standaard over code kwaliteit van de ene developer met de standaard van de andere developers geijkt. Dit zorgt ervoor dat de standaarden die door het bedrijf worden aangehouden doormiddel van de eerder aangegeven middelen ook worden geimplementeerd door de developers en deze hiervan op de hoogte blijven. (Srinivasan, 2016).

8.6.7 Dashboards

Om het scala aan reporting frameworks en regelcheckers te kunnen blijven volgen is een dashboard implementatie onmisbaar. Gegevens over de code kwaliteit zijn zinloos wanneer deze niet inzichtelijk gemaakt kan worden. Daarom is het belangrijk om Tooling bij een ontwikkelstraat in te zetten zodat de tips en verplichtingen die worden gesteld aan code kwaliteit niet gemist worden. (Srinivasan, 2016).

8.7 Onderzoek tools

Deze matrixen hebben we gebouwd en ingevuld aan de hand van de resultaten van de deelvragen. Per onderdeel van de ontwikkelstraat hebben we een aparte matrix gemaakt.
De onderdelen kunnen wij invullen door de tools uit te proberen en door te experimenteren.
Voor de volgende onderdelen gaan wij een experiment uitvoeren:

- Automated Testing
- Version control

8.7.1 Integration Tool

Tool	Dashboards	Notificatie	Integratiemogelijkheden	Build tool support
Jenkins/ Hudson	Ja	Android, Email, Google Calendar, IRC, XMPP, RSS, Twitter, Slack, Catlight, CCMenu, CCTray	Eclipse, IntelliJ IDEA, NetBeans, Bugzilla, Google Code, Jira, Bitbucket, Redmine, FindBugs, Checkstyle, PMD and Mantis, Trac, HP ALM	Maven, Ant, NAnt, shell scripts, Kundo, MSBuild, Windows batch commands
Travis	Ja	Email, Campfire, HipChat, IRC, Slack, Catlight, CCMenu, CCTray	GitHub, Heroku	Ant, Maven, Gradle
Bamboo	Ja	XMPP, Google Talk, Email, RSS, Remote API, HipChat	IntelliJ IDEA, Eclipse, Visual Studio, FishEye, Jira, Clover, Bitbucket, GitHub	MSBuild, NAnt, Visual Studio, Ant, Maven
CruiseControl	Ja	Email, CCTray	Eclipse	NAnt, Rake, Xcode, Phing, Apache Ant, Maven
CruiseControl.NET	Ja	Email, CCTray, RSS	-	MSBuild, NAnt, Visual Studio
TeamCity	Ja	Email, XMPP, RSS, IDE, SysTray	Eclipse, Visual Studio, IntelliJ IDEA, RubyMine, PyCharm, PhpStorm, WebStormm, Jetbrains Youtrack, Jira, Bugzilla, FishEye, FindBugs, PMD, dotCover, NCover	MSBuild, NAnt, Visual Studio, Duplicates finder for .NET, Ant, Maven 2-3, Gradle, IntelliJ IDEA

8.7.2 Testing: Unit Testing

Tool	Support	IDE Integrations
csUnit	.NET	Visual Studio
Visual Studio Test Professional	.NET	Visual Studio
xUnit.net	C#, F#, .NET	Visual Studio
NUnit	.NET	Visual Studio
DotCover	.NET	Visual Studio
QUnit	Javascript	WebStorm
Mocha	Javascript	Visual Studio, WebStorm
Jasmine	Javascript	WebStorm

8.7.3 Testing: Automated Testing

Tool	Script taal	Reporting	Record & playback
Selenium	Ruby, Java, PHP, Perl, Python, C#, Groovy	Ja	Ja
WatiN	C#	Ja	Ja
eggPlant	SenseTalk	Ja (eggPlant Manager)	Ja
TestComplete	VB, JS, C++, C#, Delphi	Ja	Ja
Test Studio	C#, VB.NET	Ja	Ja
Sahi	Sahi Script	Ja	Ja
Protractor	Javascript	Ja	

Opmerkingen: In deze lijst zijn zowel web en gui testing tools opgenomen.

8.7.4 Version Control

Tool	Branches	Merge conflict tools	Code review	Distributed	Populariteit	Ignore files
Git	Y	Y	External	Y	91	Y
TFS	Y	Y	Y	N	10	N
CVS	Y	Manual	N	N	0	Y
Mercurial	Y	Y	External	Y	15	Y

Voor het meten van Version control zijn een aantal tests uitgevoerd en een aantal eigenschappen zijn uit de documentatie van de tooling gehaald. Daarnaast is google trends (Google, 2016) gebruikt voor de analyse van de populariteit van het versie beheer programma.

Voor het testen van de branching functionaliteit is voor Git, TFS en Mercurial een locale installatie getest op deze functionaliteit. Deze waren allen in staat om branches aan te maken en deze consistent te bewaren om vervolgens terug te mergen. Voor CVS is hiervoor de documentatie geraadpleegd op de website van CVS (mdb, 2006). Volgens deze documentatie kan CVS in dit opzicht dezelfde functionaliteit waarmaken als de andere version control programma's.

Voor het resolven van merge conflicts zijn git, TFS en Mercurial lokaal getest. Deze tools waren allen in staat om in het geval van een merge conflict de gebruiker een overzichtelijke interactie te geven om deze op te lossen. CVS bied hier echter geen goede functionaliteit voor aan en alle conflict resolutie moet handmatig verricht worden door de developer.

Voor het uitvoeren van code reviews hebben Git en Mercurial een uitgebreide keuze aan externe tools. TFS heeft een ingebouwde code review module die goed samenwerkt met andere tooling uit het visual studio team services framework. Voor CVS zijn geen publieke code reviewing tools te vinden.

Een sterke weegfactor voor ons team was of het programma gedistribueerd kan werken. Dit stelt de developers in staat om te kunnen werken zonder een permanente server verbinding, het op kunnen stellen van meerdere remotes. Dit zorgt voor een grote verbetering in ontwikkeltijd.

Ook hebben wij een onderzoek gedaan naar de populariteit onder developers van de verschillende vcs programma's. Dit kan van grote invloed zijn wanneer andere developers het team later komen versterken of wanneer er recente informatie over het systeem gezocht moet worden op internet. Hieruit bleek dat Git een zeer populair programma is.

Ook is het van belang dat wanneer er een framework gebruikt wordt, dat de files die hierbij komen niet verplicht mee worden genomen in het versie beheer. Daarom hebben wij bij onze experimenten ook gecontroleerd wat de fuctionaliteit was om bestanden buiten het versie beheer programma te sluiten. Alle proramma's behalve TFS hadden hier een special bestand voor waarin bestanden en regels gezet konden worden om hiermee bestanden uit te zonderen. TFS kan alleen via een lange omweg specifieke bestanden uitsluiten.

8.7.5 Code Review

Tool	**VCS**	**Mogelijkheid om discussie te starten**	**Mogelijkheid om opmerkingen te plaatsen**	**Populariteit**
Gerrit	Git	Ja	Ja	Klein
Crucible	Subversion, Git, Mercurial, Perforce	Ja	Ja	Middelmatig
GitLab	Git	Ja	Ja	Middelmatig
GitHub	Git	Ja	Ja	Groot
Review Assistant	TFS	Ja	Ja	Middelmatig

Een code review is niet automatisch. Hierbij kijken de programmeurs zelf naar de code. Dit wordt gebruikt om zichtbaar problematische code te verbeteren. De menselijke fouten worden hier gevonden. Het liefst voordat de code verder wordt doorgevoerd in de ontwikkelstraat. (Huston, n.d.)

8.7.6 Static Code Analysis

We beginnen met deze code base voor C#: https://github.com/aalhour/C-Sharp-Algorithms

Voor Javascript is deze code base gebruikt:
https://github.com/FreeCodeCamp/FreeCodeCamp

Rapportage en snelheid testen wij volgens de volgende Stappen:

1. Voer de code analyse uit, meet hierbij de snelheid
2. Worden na het uitvoeren de metrics uit h9.2 weergeven?
3. Hoe wordt de uitkomst van de analyse gerapporteerd?

Tool	Rapportage	Snelheid	Support voor metrics uit H8.2
SonarQube	Uitgebreide GUI, met uitleg	32 seconden	Ja
Fxcop	Als warnings na het bouwen	11 seconden	Nee
Angular2/JS			
JSLint	Uitgebreid per regel wat er mis is	1 seconde	Nee
Google Closure Linter	Per regel wordt aangegeven wat fout is	6 seconden	Nee

Opmerkingen: SonarQube vereist een hoop opzet, maar de rapportage is een stuk duidelijker en bruikbaarder. Google Closure Linter geeft ook de mogelijkheid om style errors automatisch te fixen met ‘fixjsstyle’.

8.7.7 Build Automation

Tool	Ondersteuning voor taken	IDE-support	Automatisch dependencies downloaden	Leercurve	Archiveren versies
NAnt	Ja	Nee	Met Ivy	Hoog	Door custom taak, lastig
MSBuild	Native C# Tasks	Ja	External (nuget)	Laag	Ja
Gulp	Externe library	In-build CLI support	Ja, via npm	Laag via CLI	Externe Library
Grunt	Native JS Tasks	In-build CLI support	Ja, via npm	Laag	Externe Library

Maven en Gradle hebben wij ook onderzocht, maar de integratie met C# is erg stroef en slecht ondersteund.

MSBuild heeft als grote voordeel dat alle custom taken toegevoegd worden via C# code. Dit maakt de taken makkelijk te lezen/organiseren en versioneren. Daarnaast biedt het ook een goede intergatie met andere tooling zoals visual studio met de in-IDE command line en package managers zoals NuGet en ProGet.

(Eberhardt, 2015) Gulp kan zeer gemakkelijk geinstalleerd en gebruikt worden voor kleine apps en opstartende project met zijn command line integration. Voor grotere projecten is het lastiger opzetten en is meer kennis nodig maar daarmee kan het build proces wel versneld en versimpeld worden.

(Grunt js, 2016) Grunt is minder uitgebreid dan gulp maar kan gemakkelijker ingezet worden voor middelgrote projecten. Bijvoorbeeld de native js tasks die ook in de cli versie beschikbaar is, stellen de developer in staat om verder te gaan met de simpele versie van de tooling.

8.7.8 IDE

Tool	Gebruikers-vriendelijkheid	Debugger	Ingebouwde VCS integratie	Automated testing integratie	Deployment mogelijkheden
MonoDevelop	Goed, weinig ingebouwde shortcuts	Ja	Nee	Door een add-on	Webdeployment
Visual Studio	Goed, onhandige ingebouwde shortcuts	Ja	Ja	Ja	Veel
SharpDevelop	Goed, code generatie / templates	Ja	Door plugins	Ja	Alleen lokaal
Webstorm	Goed, gebouwd op IDEA platform	Ja	Ja	Ja	Via ingebouwde CLI
Sublime Text	Redelijk	Nee	Nee	Nee	Nee

Opmerkingen: MonoDevelop en SharpDevelop zijn duidelijk minder professioneel dan Visual Studio, en zijn hierdoor beperkter.

8.7.9 Algemene eigenschappen

Tool	**Plugins**	**Open Source**	**Documentatie**	**Prijs**	**Cross platform**	**Makkelijke configuratie**
Jenkins/ Hudson	Ja(veel)	Ja	Ja	Gratis	Ja	Ja
Travis	Ja	Ja	Ja	Gratis of betalen	Ja	Redelijk
Bamboo	Ja	Nee	Ja	Betalen	Ja	Redelijk
CruiseControl	Ja	Ja	Ja	Gratis	Ja	Ja
CruiseControl.net	Ja	Ja	Ja	Gratis	Ja	Ja
TeamCity	Ja	Nee	Ja	Gratis voor open source projecten	Ja	Redelijk

csUnit	Nee	Ja	Ja	Gratis	Windows	Ja
Visual Studio Test Professional	Ja (Visual Studio)	Nee	Ja	Betalen	Windows	Ja
WatiN	Nee	Ja	Ja	Gratis	Web	Ja
xUnit.net	Nee	Ja	Ja	Gratis	Windows	Ja
NUnit	Nee	Ja	Ja	Gratis	Windows	Ja
DotCover	Ja (ReSharper)	Nee	Ja	Betalen (ReSharper)	Windows	Ja
QUnit	Ja	Nee	Ja	Gratis	Ja	Ja
Mocha	Nee	Ja	Ja	Gratis	Ja	Ja

Tool	Plugins	Open Source	Documentatie	Prijs	Cross platform	Makkelijke configuratie
Jasmine	Nee	Ja	Ja	Gratis	Ja	Redelijk
Selenium	Voor Selenium IDE	Ja	Ja	Gratis	Ja	Ja
eggPlant Functional	Nee	Nee	Ja	Betalen	Ja	Ja
TestComplete	Nee	Nee	Ja	Betalen	Windows, Android, iOS	Ja
TestStudio	Nee	Nee	Ja	Betalen	Windows, Web	Ja
Sahi	Nee	Ja	Ja	Gratis of betalen	Web	Redelijk
ProTractor	Ja	Ja	Ja	Gratis	Ja	Redelijk

Git	Ja	Ja	Ja	Gratis	Ja	Ja
TFS	Ja	Nee	Ja	Gratis of betalen	Alleen Windows	Nee
CVS	Nee	Ja	Nee	Gratis	Ja	Zeer lastig
Mercurial	Ja	Ja	Ja	Gratis	Ja	Redelijk

Gerrit	Ja	Nee	Ja	Gratis	Ja	Ja
Crucible	Ja	Nee	Ja	Betalen	Ja	Ja
GitLab	Nee	Nee	Ja	Gratis	Ja	Ja
GitHub	Nee	Nee	Ja	Gratis of betalen	Ja	Ja
Review Assistant	Nee	Nee	Ja	Betalen	Windows	Ja

SonarQube	Ja	Ja	Ja, Veel	Gratis of betalen	Cross platform	Nee
Fxcop	Nee	Nee	Ja	Gratis	Windows	Ja
JSLint	Nee	Nee	Ja	Gratis	Ja	Ja
Google Closure Linter	Nee	Nee	Niet veel	Gratis	Ja	Ja

NAnt	Nee	Ja	Matig	Gratis	Windows	Nee
MSBuild	Nee	Ja	Matig	Gratis	Windows	Redelijk
Gulp	No	Ja	Matig	Gratis	Cross platform	Redelijk
Grunt	No	Ja	Ja	Gratis	Cross platform	Ja

Tool	**Plugins**	**Open Source**	**Documentatie**	**Prijs**	**Cross platform**	**Makkelijke configuratie**
MonoDevelop	Ja	Ja	Weinig	Gratis	Cross platform	Ja
SharpDevelop	Ja	Ja	Weinig	Gratis	Cross platform	Ja
Visual Studio	Ja	Nee	Ja, Veel	Gratis of betaald	Windows	Ja
Webstorm	Ja	Nee	Ja	Betaald (Gratis voor studenten)	Cross platform	Ja

9 Conclusie en discussie

Om de onderzoeksvraag "*Hoe kunnen we in een softwareproject met behulp van een ontwikkelstraat een hogere codekwaliteit realiseren ten op zichten van een softwareproject zonder ontwikkelstraat?"* te beantwoorden maken wij gebruik van de deelresultaten.

We kunnen een hogere codekwaliteit realiseren met een ontwikkelstraat door voor de specifieke toepassing een aantal tools samen te voegen die elk een toevoegende waarde hebben voor het verbeteren van de code kwaliteit. We bouwen de ontwikkelstraat op uit de volgende onderdelen: automatische integratie, automatische builds, automatische tests, versiebeheer, statische code analyse en code reviews.

Als deze onderdelen goed geconfigureerd zijn dragen zij allemaal bij aan de codekwaliteit door het versnellen en automatiseren van veelvoorkomende handelingen en het uitfilteren van menselijke fouten. Dit is zonder het gebruik van deze tools, ook wel ontwikkelstraat, een stuk lastiger. Zonder gebruik van een ontwikkelstraat zou dit meer tijd kosten en kunnen er fouten optreden in processen waarin niks is veranderd.

Er wordt op allerlei gebieden tijd bespaart. Het belangrijkste is, doordat de codekwaliteit hoger is en de ontwikkelstraat wordt toegepast, is het makkelijker om de codebase uit de breiden of wijzigingen hierin te maken. Het gebruik van een ontwikkelstraat vereist wel een bepaalde werkwijze. Bij het implementeren van nieuwe features of wijzigen van bestaande features moet de code door de ontwikkelstraat heen op een bepaalde volgorde. Dit levert hoge codekwaliteit door een constante en directe controle van de code.

9.1 Discussie

Dit antwoord komt voor een groot deel overeen met de gestelde hypothese.

"Wij verwachten dat een ontwikkelstraat met onderdelen specifiek gekozen voor de toepassing ervoor zorgt dat de codekwaliteit hoger is, door alle hulpmiddelen die de ontwikkelstraat biedt, en door de werkwijze die gehanteerd wordt".

Bij het ontwikkelen met een ontwikkelstraat moet er consequent gebruik gemaakt worden van de tools, anders werkt het niet optimaal. Dat is de werkwijze die van belang is. Het is een combinatie van automatische tools en het goed gebruikmaken hiervan.

De resultaten die wij gevonden hebben bij het onderzoeken zijn voor ons niet onverwacht.

10 Evaluatie

Het onderzoek heeft een aantal goede en zwakke punten. Een zwak punt van het onderzoek is dat de resultaten sterk afhankelijk zijn van de omgeving van het onderzoek. Dit zorgt ervoor dat het onderzoek voor elke afzonderlijke “case” apart uitgevoerd moet worden om tot een nieuwe conclusie te komen of een ontwikkelstraat wel of niet goed te gebruiken is.

Bij het onderzoeken hebben wij weinig geëxperimenteerd. De resultaten zijn hierdoor vrijwel allemaal uit literatuuronderzoek opgesteld. Hierdoor hebben wij een conclusie getrokken op basis van literatuur. Dit zou mogelijk in de toekomst nog voor verassingen zorgen. De samenstelling van tools kan mogelijk bij het uitvoeren van de opdracht worden herzien.

Wij hebben ook geleerd dat het belangrijk is om een concrete onderzoeksvraag te formuleren. We hebben voor dit onderzoek een te abstracte onderzoeksvraag geformuleerd. Hierdoor was het lastig om deze goed, concreet, te beantwoorden.

Sterke punten van het onderzoek vinden wij dat we een scala aan tools onderzocht hebben. Dit is niet direct nodig voor het beantwoorden van de onderzoeksvraag, maar geeft wel een beter beeld van de ontwikkelstraat en hoe deze het best geïmplementeerd kan worden. Deze informatie kan worden gebruikt bij de opdracht die wij hierna gaan uitvoeren.

11 Aanbevelingen

Op basis van de resultaten van het bovenstaande onderzoek, zullen we in dit hoofdstuk een samenstelling maken van een ontwikkelstraat.

11.1 Integration Tool

Als integration tool raden wij om Jenkins te gebruiken. Redenen hiervoor zijn dat er veel documentatie beschikbaar is en deze tool is ook het meest populair tussen de integration tools. Dit betekent dat het support die je kan krijgen voor deze tool veel groter is vergeleken met andere tools, en er zijn veel tools die met Jenkins geïntegreerd kunnen worden. Voor Jenkins zijn er ook een veel aantal plugins beschikbaar die je kan gebruiken. Als laatste punt is Jenkins een open source tool en gratis.

11.2 Version Control

Als eerste moet een ontwikkelstraat een repository hebben waarin de code opgeslagen kan worden en waarin de verschillende versies van de code op een consistente, overzichtelijke en snelle manier kan worden beheerd. Hiervoor hebben wij onderzoek gedaan naar tooling zoals version control software die ons hierbij kan helpen. Uit ons onderzoek is gebleken dat Git hiervoor de beste keuze is gezien zijn gedistribueerde opbouw en dat deze gratis gebruikt kan worden.

11.3 IDE

Vervolgens wordt gekeken naar tooling die gebruikt kan worden voor het schrijven/aanpassen van code te versnellen en meer overzicht te creëren voor de developer. Hiervoor hebben wij verschillende Integrated Development Environments vergeleken en zijn tot de conclusie gekomen dat Visual Studio het best kan worden gebruikt voor het ontwikkelen van c# projecten. Dit omdat een product is van microsoft en daardoor zeer sterke documentatie heeft en een sterke integratie kan maken tussen andere tooling die veelal wordt gebruikt bij C#.

Voor het werken met Angular2 raden wij aan om WebStorm te gebruiken als IDE. Dit ook omdat er een groot bedrijf achter de software zit dat veel documentatie en support biedt en veel integraties biedt met tooling en services die vaak voorkomen in Angular en andere javascript projecten.

11.4 Automated Testing

Voor unit testing van C# raden wij aan om NUnit te gebruiken. De keuze is hierop gevallen gezien deze unit testing tool een directe integratie heeft met Visual Studio. Daarnaast heeft het ook de voordelige combinatie dat het open source is, goed gedocumenteerd en gratis onbeperkt te gebruiken is. Voor het unit-testen van JavaScript raden wij het gebruik van Mocha aan. Mocha is makkelijk te integreren met de IDE, en is makkelijk te configureren.

Voor het automatisch testen (gui-tests) van Angular2 (JavaScript) raden wij het gebruik van Protractor aan. Protractor is open-source en heeft een grote gebruikersbasis waardoor er veel oplossingen op het internet te vinden zijn.

11.5 Code Review

Voor het de code review tooling raden wij aan om gebruik te maken van GitHub. Dit is omdat Github gratis is voor een onbeperkt aantal projecten en code, het populair is onder developers en het is gericht op de eerder gekozen version control Git. Daarnaast krijg je ook plugins tot je beschikking zoals Zenhub welke voor een goed overzicht zorgt van de huidige projectstatus. Dit zorgt ervoor dat alle developers zonder moeite code reviews kunnen sturen en ontvangen en dit kunnen toepassen in de ontwikkelstraat.

11.6 Static Code Analysis

Voor statische code analyse raden wij SonarQube aan voor C# en SonarLint voor Angular2 / JavaScript. SonarQube is gekozen vanwege zijn uitgebreide en overzichtelijke GUI en documentatie die daarin beschikbaar is. Daarnaast is het in deze tool ook makkelijk om eigen regels toe te passen en deze vervolgens te beheren.

JSLint is gekozen omdat dit open-source is en er veel JavaScript programmeurs achter de regels die JSLint stelt staan. JSLint kan helaas niet complexiteit en andere metrics meten, maar kijkt vooral op syntax niveau.

11.7 Build Automation

Voor het automatiseren van de C# builds raden wij MSBuild aan. Deze keuze is gemaakt omdat deze een lage leercurve heeft, wat deze geschikt maakt voor de korte projectduur. Daarnaast heeft het een sterke integratie met Visual studio, de eerder gekozen IDE. Ook biedt het de mogelijkheid om build versies te archiveren wat een goede toevoeging is aan de ontwikkelstraat.

12 Suggesties voor verder onderzoek

Tijdens het onderzoek zijn wij erachter gekomen dat het erg lastig is om een goede manier van meten te hanteren voor deze vrij abstracte concepten. Codekwaliteit is ook deels subjectief, waardoor het niet door iedereen op dezelfde manier kan worden gemeten.

Hier zou een andere professionele partij ook een onderzoek naar kunnen doen om zo een standaard voor het meten van codekwaliteit op te stellen.

Daarnaast is het opzetten van een ontwikkelstraat en de onderdelen die in de ontwikkelstraat zitten makkelijker door een duidelijke context te schetsen. Als duidelijk is waar de ontwikkelstraat voor dient kan er een gerichter onderzoek worden uitgevoerd.

Voor andere programmeertalen dan C# en Angular2 raden wij aan om eenzelfde onderzoek uit te voeren om zo een ontwikkelstraat te bouwen.

13 Bibliografie

(sd). Opgehaald van http://www.csharptools.com/

(sd). Opgehaald van Monodevelop: http://www.monodevelop.com/documentation/feature-list/

Allan Kelly. (2010, 06 03). *Things to do to improve code quality*. Opgehaald van DZone: https://dzone.com/articles/things-do-improve-code-quality

Atlassian. (sd). Opgehaald van Atlassian Crucible: https://www.atlassian.com/software/crucible

Build Tool. (sd). Opgehaald van Techopedia: https://www.techopedia.com/definition/16359/build-tool

Continuous Delivery Tools List. (2011). Opgehaald van Stelligent: https://stelligent.com/2011/03/23/continuous-delivery-tools-list/

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

Devart. (sd). Opgehaald van Devart Review Assistant: https://www.devart.com/review-assistant/

Duvall, P. M. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk.*

Eberhardt, C. (2015, 12 24). *CREATING AN ANGULAR 2 BUILD WITH GULP, TSLINT AND DEFINITELYTYPED*. Opgehaald van Scott Logic: http://blog.scottlogic.com/2015/12/24/creating-an-angular-2-build.html

Farley, D., & Humble, J. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.*

Fernandes, T. (2013, March). *What's in a Software Factory?* Opgehaald van Bitmaker Software: http://www.bitmaker-software.com/whats-in-a-software-factory

Fowler, M. (2006). *Continuous Integration*. Opgehaald van Martin Fowler: http://martinfowler.com/articles/continuousIntegration.html

Fowler, M. (2013, May). *ContinuousDelivery*. Opgehaald van Martin Fowler: http://martinfowler.com/bliki/ContinuousDelivery.html

Fxcop. (sd). Opgehaald van Fxcop: https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx

gerrit. (sd). *Gerrit review*. Opgehaald van Google Source: https://gerrit-review.googlesource.com/Documentation/

Gitlab. (2014, 09 29). Opgehaald van Gitlab: https://about.gitlab.com/2014/09/29/gitlab-flow/

Google. (2016, 10 12). *Google Trends*. Opgehaald van Google Trend analytics: https://www.google.com/trends/explore?q=%2Fm%2F05vqwg,%2Fm%2F02rvgkm,%2Fm%2F08441_,%2Fm%2F09d6g

Grunt js. (2016, 10 25). *Configuring Tasks*. Opgehaald van gruntjs: http://gruntjs.com/configuring-tasks

Huston, T. (sd). *What is Code Review*. Opgehaald van Smart Bear: https://smartbear.com/learn/code-review/what-is-code-review/

icsharpcode.net. (sd). Opgehaald van ICSharp: http://www.icsharpcode.net/OpenSource/SD/Features.aspx

Integrated Development Environment (IDE). (sd). Opgehaald van Techopedia: https://www.techopedia.com/definition/26860/integrated-development-environment-ide

ISO/IEC 25010:2011 - Systems and software engineering. (2011). Opgehaald van www.iso.org: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=35733

Jack Greenfield, K. S. (2004). *Software Factories - Assembling, Applications with Patterns, Models, Framework, and Tools.*

Martin, R. C. (2008). *Clean Code: A handbook of agile software craftmanship.* United States: Pearson.

mdb. (2006, 06 03). *CVS - Concurrent Versions System*. Opgehaald van Non-GNU: http://www.nongnu.org/cvs/

MonoDevelop Documentation. (sd). Opgehaald van MonoDevelop: http://www.monodevelop.com/documentation/creating-aspnet-projects/

NAnt C# meets Ant. (sd). Opgehaald van NAnt: http://nant.sourceforge.net

Rouse, M. (sd). *SearchWinDevelopment*. Opgehaald van static analysis (static code analysis): http://searchwindevelopment.techtarget.com/definition/static-analysis

Rüdger, L., Jonas, L., & Welf, L. (2008). *Comparing Software Metric Tools.* School of Mathematics and Systems Engineering.

Software Test Automation Tools. (sd). Opgehaald van Testingtools: http://www.testingtools.com/test-automation/

SonarQube. (sd). Opgehaald van SonarQube.org: http://docs.sonarqube.org/display/SONAR/Get+Started+in+Two+Minutes

Srinivasan, B. N. (2016). *What Tools Do You Need for Continuous Delivery?* Opgehaald van DZone: https://dzone.com/articles/101-tools-for-continuous-delivery

Weber, H. (1997). *The Software Factory Challenge.*

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.

14 Bijlagen

14.1 Bijlage A: Vragen interviews

De volgende vragen hebben wij gesteld aan een aantal professionals:

1. Wat betekent codekwaliteit volgens u?
2. Op wat voor manier realiseert u hoge codekwaliteit ?
3. Hoeveel waarde hecht u aan codekwaliteit ?
4. Hoe meet u codekwaliteit?
5. Wat voegt een hoge codekwaliteit volgens u toe aan software?
6. Maakt u gebruik van een ontwikkelstraat?
7. Welke onderdelen bevat uw ontwikkelstraat?
8. Zijn er wijzigingen geweest binnen de productie/ontwikkelstraat ten goede van de codekwaliteit? Wat waren de wijzigingen
9. Zijn er volgens u nadelen aan het gebruik van een ontwikkelstraat? Wat zijn volgens u de nadelen?

14.2 Bijlage B: Response interviews

14.2.1 Response Aiko Mastboom (Evalytics, Vrendly)

Vraag: Wat betekent codekwaliteit volgens u?
Antwoord: *Code kwaliteit is een samenspel tussen een aantal vershillende aspecten aan code:*

- *uniforme syntax verhoogt de leesbaarheid voor andere ontwikkelaars*
- *unit tests verhogen de inzichtelijkheid van kleine modules code*
- *comments op lastig te doorzien plaatsen verhogen leesbaarheid en begrip voor andere ontwikkelaars.*

Vraag: Op wat voor manier realiseert u hoge codekwaliteit ?
Antwoord:

- *Code testen op afgesproken syntax (lint)*
- *Gezamenlijke code reviews.*
- *Automatische draaien van unit / end-to-end tests door een continues integratie straat.*

Vraag: Hoeveel waarde hecht u aan codekwaliteit ?
Antwoord: *Veel, aangezien we hier een hoge wisseling van ontwikkelaars hebben (veel stagiaires) is het voor ons erg belangrijk dat iedereen (en nieuwe mensen) snel met elkaars code aan de slag kunnen.*

Vraag: Hoe meet u codekwaliteit?
Antwoord:
Automatisch:

- *syntax / formatting checks (moet slagen)*
- *unit tests (moeten slagen)*
- *code coverage (moet minimaal gelijk blijven na toevoegen van features)*
- *e2e tests op applicatie nivo (moeten slagen)*

Handmatig:

- *code reviews*

Vraag: Wat voegt een hoge codekwaliteit volgens u toe aan software?
Antwoord:

- *leesbaarheid,*
- *testbaarheid,*
- *soms herbruikbaarheid,*
- *sneller inwerken van nieuwe mensen*

Vraag: Maakt u gebruik van een ontwikkelstraat?
Antwoord: *Ja, wij hebben een Jenkins server ingericht om code automatisch van development (git push) naar productie (docker image) te brengen.*

Vraag: Welke onderdelen bevat uw ontwikkelstraat?
Antwoord:
DEVELOPMENT:
Met behulp van docker containers werken we hier zo dicht mogelijk tegen productie omgeving aan. Jenkins CI build, test, packaged en e2e tests de software. (alle stappen die CI zet kan ontwikkelaar ook op zijn eigen machine uitvoeren)

TEST:
Deze missen we nog (nog geen directe noodzaak voor gehad)

STAGING:
Front en backend gedeployed in productie (like) omgeving. (ge-sandboxed zodat er bijv geen mails naar klanten gaan)

LIVE:
Zelfde als staging maar live voor klanten.

Vraag: Zijn er wijzigingen geweest binnen de productie/ontwikkelstraat ten goede van de codekwaliteit? Wat waren de wijzigingen?
Antwoord: *Doordat we de productie omgeving dichter bij de developer hebben gehaald zijn deze zich bewuster geworden van de uitdaging die zich tijdens deployment voordoen. Hierdoor is de software configureerbaarder (per stage) geworden en kunnen we sneller schakelen tussen de verschillende stages.*

Vraag: Zijn er volgens u nadelen aan het gebruik van een ontwikkelstraat? Wat zijn volgens u de nadelen?
Antwoord: *Het vraagt een behoorlijke portie discipline en toch wel wat (voor)kennis om het geheel te begrijpen en goed te gebruiken.*

14.2.2 Response Kah Loon Yap, quality control manager Exact

Vraag: What does "Code Quality" mean to you?
Antwoord: *Code Quality says something about how easy it is to understand and to change (maintain) code, and not just how easy it is for yourself, but especially for others.*

Vraag: How do you realize high code quality?
Antwoord: *Writing small methods, stick to the S.O.L.I.D. principles and unit test my code.*

Vraag: How much do you value code quality?
Antwoord: *I value code quality very highly, because I've seen where a lack of code quality can lead to: a system that is hard to change, which impacts your productivity and hinders innovation.*

Vraag: How do you measure code quality?
Antwoord: *I use several metrics for measuring code quality: method length, class length, class coupling and (unit test) code coverage.*

Vraag: What does high code quality add to the software according to you?
Antwoord: *It gives you the agility to adapt the software over time, so that the software really is 'soft' It also adds reliability and confidence to the delivered solution.*

Vraag: Have there been any changes to your production/software factory to augment the code quality?
Antwoord: *For every new commit in our source repositories, we use SonarQube to automatically detect code smells and to measure code coverage.*

14.2.3 Response Peter van Vliet (Uit de hoogte)

Vraag: Wat betekent codekwaliteit volgens u?
Antwoord: *Code is van hoge kwaliteit als deze leesbaar, begrijpbaar, uitbreidbaar en onderhoudbaar is. Een (ervaren) ontwikkelaar die onbekend is met een systeem zou zonder al te veel moeite de code moeten kunnen doorgronden en basis onderhoud kunnen uitvoeren (debugging).*

Vraag: Op wat voor manier realiseert u hoge codekwaliteit ?
Antwoord: *Toepassen van clean-code principes en code-reviews van minder triviale stukken code.*

Vraag: Hoeveel waarde hecht u aan codekwaliteit ?
Antwoord: *Steeds meer. Wij leveren inmiddels bijna al onze code op aan klanten die hier intern mee uit de voeten moeten kunnen. Daarnaast hebben we steeds meer te maken met Escrow-overeenkomsten.*

Vraag: Hoe meet u codekwaliteit?
Antwoord: *Hier hebben we nog geen tool voor draaien (op een later moment willen we SonarQube inzetten). Nu zijn we nog afhankelijk van onderlinge reviews.*

Vraag: Wat voegt een hoge codekwaliteit volgens u toe aan software?
Antwoord:

1. Minder kans op programmeerfouten.

2. Vereenvoudiging van foutopsporing.

3. Vereenvoudiging van het schrijven van tests (unit/integratie).

4. Eenvoudiger uit te breiden / wijzigen.

5. Eenvoudiger over te dragen.

Vraag: Maakt u gebruik van een ontwikkelstraat?
Antwoord: *Ja (op sommige punten nog een beetje houtje-touwtje).*

Vraag: Welke onderdelen bevat uw ontwikkelstraat?
Antwoord:

Individueel niveau

Technologie: Java, HTML5

Tools: Eclipse/Netbeans IDE, unit-testtools

Teamniveau

Source control: Git (gitflow)

Methodiek: Scrum

Issue tracking: Jira

Communicatie / messaging: Slack

Organisatieniveau

Issue tracking: Jira

Urenregistratie: Jira

Deployment: Capistrano

Release management: OTAP omgeving

Vraag: Zijn er wijzigingen geweest binnen de productie/ontwikkelstraat ten goede van de codekwaliteit? Wat waren de wijzigingen?
Antwoord: *Neen, deze moeten nog komen.*

Vraag: Zijn er volgens u nadelen aan het gebruik van een ontwikkelstraat? Wat zijn volgens u de nadelen?
Antwoord: *Voor kleine projecten levert een ontwikkelstraat veel overhead. Voor grote(re) projecten zie ik niet zo snel nadelen (indien de straat naar behoefte is ingericht, wat soms nog een uitdaging kan zijn).*

14.3 Bijlage C: Quotes Clean Code

Bjarne Stroustrup, inventor of C++ and author of *The C++ Programming Language*
I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well.

Grady Booch, author of Object Oriented Analysis and Design with Applications
Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control.

"Big" Dave Thomas, founder of OTI, godfather of the Eclipse strategy
Clean code can be read, and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone.

Ron Jeffries, author of Extreme Programming Installed and Extreme Programming Adventures in C#
- *Runs all the tests;*
- *Contains no duplication;*
- *Expresses all the design ideas that are in the system;*
- *Minimizes the number of entities such as classes, methods, functions, and the like.*