Datavisualisatie ORTEC Sports

Wiechert Toonstra
Studentnummer: 15054713

ORTEC
www.ortec.com

Auteur
Wiechert Toonstra
Studentnummer: 15054713

Opdrachtgever
Ruud van der Knaap
ORTEC Sports B.V.

Bedrijfsmentor ORTEC Sports
Arjan Peters

Afstudeerbegeleider
Tim Cocx

Opleiding
HBO-ICT
Differentiatie: Software Engineering
Haagse Hogeschool

Voorwoord

Dit document is geschreven in het kader ter afronding van de vierjarige studie HBO-ICT aan de Haagse Hogeschool. De afgelopen 17 weken heb ik gewerkt aan de ontwikkeling van een library waarmee er sneller en herbruikbare datavisualisaties gecreëerd kunnen worden. Dit project is uitgevoerd in opdracht van ORTEC Sports B.V.

Deze afstudeeropdracht en de totstandkoming van dit afstudeerverslag hadden niet kunnen ontstaan zonder een aantal personen. Tot hen wil ik een dankwoord richten voor hun medewerking en bereidbaarheid voor het toewijden van hun tijd.

Allereerst gaat dank uit naar ORTEC B.V., met in het bijzonder ORTEC Sports, waar ik de mogelijkheid heb gekregen om de afstudeeropdracht te uit te voeren. Binnen de afdeling gaat mijn speciale dank uit naar de heer A. Peters, voor het op zich nemen van de verantwoordelijkheid van bedrijfsmentor. Ook gaat binnen de afdeling speciale dank uit naar de heer F. van Buel, voor het bijstaan in het vinden van de oplossing voor vooral technische vraagstukken.

Wiechert Toonstra
Zoetermeer, mei 2019

Inhoudsopgave

1 Inleiding

Onderdeel van de opleiding HBO-ICT is in het laatste jaar van de studie een afstudeeropdracht te volgbrengen. Aangezien ik voor aanvang van de afstudeerperiode al werkzaam was bij ORTEC Sports, is er gekozen om hier de afstudeeropdracht te volbrengen.

Dit afstudeerverslag heeft als doel om de examinatoren en gecommitteerde inzicht te geven in hoe de afstudeerperiode is verlopen. Dit inzicht zorgt er voor dat er over het uiteindelijke afstudeertraject een beoordeling geformuleerd kan worden t.b.v. het afstudeerproces.

Dit afstudeerverslag begint met het beschrijven van de organisatie waarin de afstudeeropdracht volbracht is. Vervolgens in hoofdstuk 3 wordt een definitie van de opdracht geformuleerd. In hoofdstuk 4 wordt een aanpak beschreven van hoe er gewerkt wordt naar een eindresultaat. Vervolgens wordt ingegaan op hoe het proces van requirements ontlokken, analyseren en documenteren is verlopen. In hoofdstuk 6 wordt de implementatie van het softwareproduct besproken. Hoofdstuk 7 bevat een beschrijving van hoe de implementatietest en unit tests zijn opgesteld en verlopen. Vervolgens in hoofdstuk 8 en 9 wordt er geëvalueerd op zowel het product als het proces. Allerlaatst wordt er nog ingegaan op de bewijsvoering van de beroepstaken die van tevoren in het afstudeerplan zijn opgesteld.

Om de leesbaarheid te waarborgen, is er aan het einde van het document een verklarende begrippenlijst toegevoegd.

2 Organisatie

2.1 Organisatiebeschrijving

ORTEC is een aanbieder van optimalisatiesoftware en analyseoplossingen. Het bedrijf is in 1981 opgericht door 5 studenten die econometrie studeerde. De diensten en producten die ORTEC biedt, zijn verdeeld over verschillende divisies, die overeenkomen met de marktsegmenten waarbinnen ORTEC actief is.

ORTEC Optimization Technology ontwikkelt zelfstandige en op maat gemaakte planningssoftware-en systemen. Hierbij optimaliseert ORTEC alles van rit- en routeplanning tot de belading van voertuigen. De software geeft informatie en overzichten van dit soort activiteiten, zodat er een beter inzicht komt in de prestaties van een bedrijf.

ORTEC consulting levert geavanceerde analyse-en optimalisatieoplossingen zoals dynamische prijsstellingen, supply-chain management, vraagvoorspellingen, kostenramingen voor klanten zoals KLM.

2.2 ORTEC Sports

ORTEC Sports is een aparte B.V. binnen ORTEC waar de afstudeeropdracht zal worden volbracht. Op deze afdeling wordt door analisten data verzameld tijdens sportwedstrijden. Het gaat hier om sporten zoals voetbal, hockey en volleybal. Dit zijn evenementen die tijdens een wedstrijd plaatsvinden, zoals passes of schoten. Deze data dient als basis voor de diensten die ORTEC Sports aanbiedt.

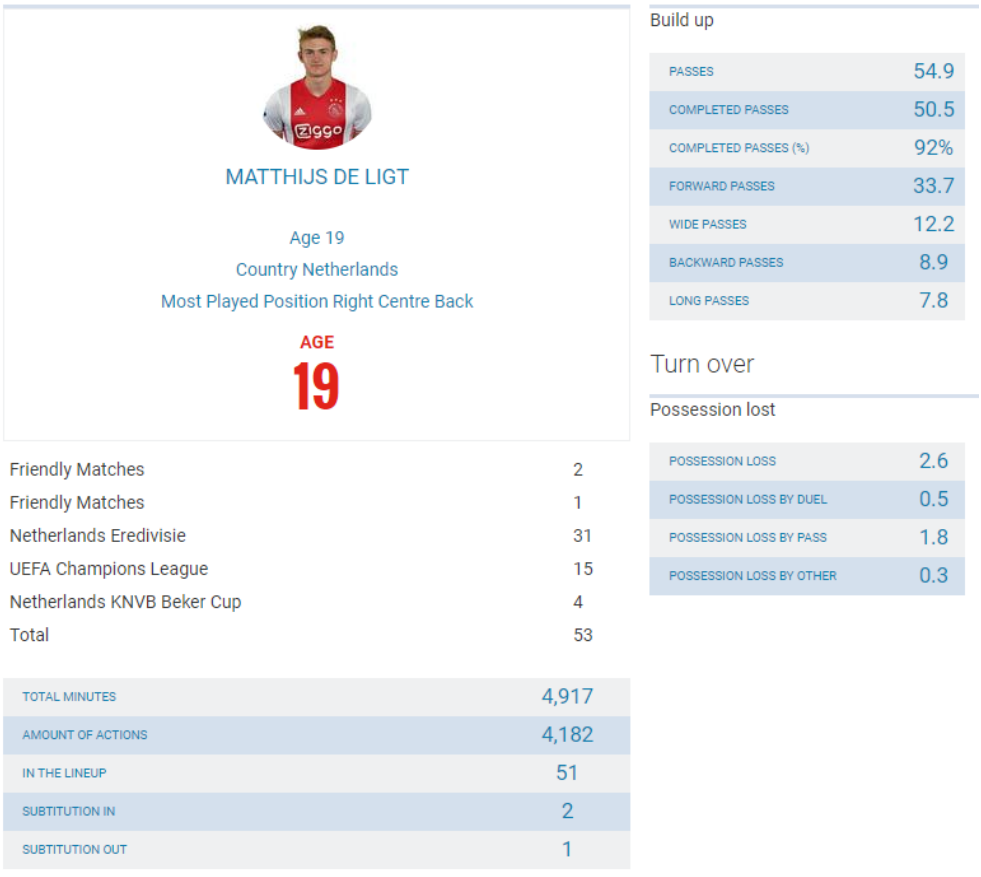
Nadat deze data verzameld is, worden er een softwareproducten mee ontwikkeld. Het gaat hier om analyses en overzichten van bepaalde wedstrijden of spelers. Zo kan er bijvoorbeeld een analyse aangeboden worden van de laatste wedstrijd van Ajax op basis van statistieken. Op deze manier kan de effectiviteit van een team of speler meetbaar worden gemaakt en dus ook worden verbeterd.

Naast het ontwikkelen van software producten, gebruikt ORTEC Sports ook de verzamelde data om te verkopen aan consultancy bedrijven in de gokmarkt. Deze bedrijven berekenen op basis van de geleverde data bepaalde kansen voor sportwedstrijden. Hierdoor kan statistisch bepaald worden hoe groot de kans is dat een bepaald team de volgende wedstrijd wint of verliest. Deze statistische kansen gebruiken grote gokkantoren voor het wedden op sportwedstrijden.

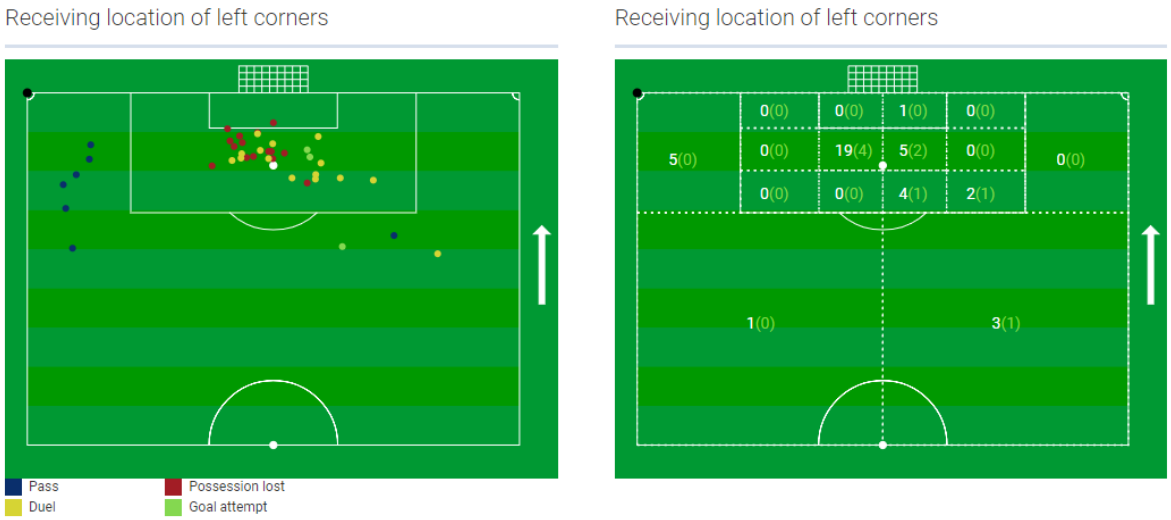
2.3 Producten en klanten

Een van de belangrijkste softwareproducten die ORTEC Sports aanbiedt, is een softwareproduct genaamd de Pro Portal. In dit product worden analyses en datavisualisaties aangeboden op basis van de verzamelde data. Het gaat hier bijvoorbeeld pass accuraatheid van een bepaalde speler in een bepaalde wedstrijd of wedstrijden. Een voorbeeld hiervan is te zien in figuur 1, waar een analyse van Ajax-aanvoerder Matthijs de Ligt is gemaakt.

Figuur 1: Pro Portal analyse



Figuur 2: Visualisatie van corners



Een ander voorbeeld is te zien in figuur 2, waar een visualisatie wordt aangeboden van corners die van de linker kant getrapt zijn.

Andere belangrijke softwareproducten die ORTEC Sports aanbiedt:

- **Digital Match Form**: digitalisatie van wedstrijdformulieren. Geen fysieke administratie meer van te spelen of gespeelde wedstrijden, maar alles digitaal.
- **Media Portal**: biedt visualisaties en analyses aan die geëxporteerd kunnen worden, zodat deze gebruikt kunnen worden door clubs op hun websites of op beeldschermen in hun stadions.
- **LiveWidget**: standen en statistieken van wedstrijden die live worden gespeeld.
- **GA (Generation Adidas) Cup-app**: communicatiemiddel tijdens de GA Cup. De GA Cup is een jaarlijks groot toernooi van professionele jeugdteams, georganiseerd door de MLS (Major League Soccer). In de GA Cup-app worden live standen en statistieken aangeboden tijdens het toernooi.

Enkele belangrijke klanten die ORTEC Sports bedient met hun producten of consultancy:

- Professionele clubs: o.a. Ajax, PSV, jeugdopleidingen MLS (Major League Soccer), Sparta.
- Media: Telegraaf, AjaxMedia
- Nevobo (Nederlandse Volleybal Bond)
- Consultancy: KNSB (Koninklijke Nederlandse Schaats Bond)

2.4 Betrokkenen

Betrokkenen binnen ORTEC Sports bij het afstudeertraject zijn de volgende personen:

Tabel 1: Betrokkenen afstudeeropdracht

Persoon	***Rol***
Ruud van der Knaap	Opdrachtgever
Arjan Peters	Begeleider/bedrijfsmentor
Bertus Talsma	Data Scientist

2.5 Technieken

Binnen ORTEC worden de volgende technieken en frameworks gehanteerd:

Tabel 2: Technieken

Onderdeel	***Techniek/hulpmiddel***
Raamwerk	Angular 7
Versiebeheer	Perforce
Distributie	NPM
Programmeertalen	HTML, CSS, TypeScript
Afhankelijkheden	D3.js
Testen	Jasmine

Er wordt gewerkt met het framework Angular. Dit is een front-end framework dat gebruikt wordt binnen ORTEC voor alle softwareproducten. Er is dan ook gekozen voor de meest recente versie van Angular, namelijk versie 7. Binnen dit framework worden de talen HTML, CSS en TypeScript gehanteerd.

Perforce[1] is een versiebeheersysteem, waarin veel van de producten van ORTEC beheerd worden.

Voor de distributie van de library is gekozen voor een private NPM. Dit is geen norm binnen ORTEC, maar wel een veelgebruikte package manager door web ontwikkelaars.

D3.js[2] is een veelgebruikte JavaScript library, die gebruikt wordt om data visualisaties mee te creëren. Deze library dient als basis voor de afstudeeropdracht.

Er worden Unit Testen geschreven met behulp van het testing framework Jasmine[3]. Dit is een framework wat gehanteerd wordt binnen ORTEC.

[1] https://www.perforce.com/solutions/version-control
[2] https://d3js.org/
[3] https://jasmine.github.io/

3 Opdrachtdefinitie

3.1 Aanleiding

ORTEC Sports verzamelt grote hoeveelheden (voetbal)data. Nadat deze data verzameld is, worden hier producten mee gebouwd om een bepaalde analyse aan te bieden. Een voorbeeld hiervan is de pass precisie van een speler over een bepaalde wedstrijd. Dit soort analyses worden op dit moment voornamelijk door ORTEC Sports aangeboden in tabellen. Er zijn bijvoorbeeld niet veel interactieve grafieken of visualisaties waar de data getoond wordt.

3.2 Probleemstelling

Momenteel bestaan er in sommige producten van ORTEC Sports al een implementatie van een aantal interactieve grafieken of visualisaties. Deze visualisaties worden keer op keer opnieuw geïmplementeerd, wat zorgt voor veel duplicaat code en minder onderhoudbare producten. Het probleem is dat deze visualisaties niet kunnen worden hergebruikt voor verschillende projecten met verschillende datasets. Ook is ondervonden dat de implementatie van zo'n grafiek vaak veel ontwikkelingstijd kost, door de vaak moeilijke configuratie.

3.3 Doelstelling

Het doel van deze afstudeeropdracht is om een library te bouwen die in verschillende projecten met verschillende datasets hergebruikt kan worden om datavisualisaties te maken. Deze library moet het makkelijker maken om bepaalde datavisualisaties te integreren in huidige en toekomstige projecten. Ook moet de library zich lenen voor uitbreidbaarheid, zodat er in de toekomst makkelijk nieuwe visualisaties aan toegevoegd kunnen worden.

3.4 Eindresultaat

Aan het einde van de afstudeerperiode worden er een aantal producten opgeleverd, namelijk:

- Requirements document
- Ontwikkelde library
- Testdocument

3.4.1 Requirements document

Tijdens de afstudeerperiode wordt er gewerkt aan het verzamelen en documenteren van de requirements. Deze requirements worden vastgelegd in een requirements document en dienen als basis voor de ontwikkeling van library.

3.4.2 Ontwikkelde library

De afstudeeropdracht gaat een product opleveren wat in de toekomst hergebruikt kan worden om bepaalde datavisualisatie te implementeren. Uit gesprekken met belanghebbenden worden de exacte requirements achterhaald en vastgelegd in een requirements document. Vanuit hier wordt besloten wat de library exact aan functionaliteit moet gaan bevatten.

Vanuit de opdrachtgever is er niet een harde eis bepaald voor welke visualisaties de library aan het eind van de afstudeerperiode moet bevatten. Er wordt gekeken naar wat haalbaar is binnen de tijdsperiode. Het is vooral belangrijk dat de library zich leent voor uitbreiding, zodat visualisaties die nog niet gemaakt zijn, in de toekomst makkelijk gemaakt kunnen worden.

Omdat datavisualisatie een veelvoorkomend proces is binnen ORTEC Sports, maar ook binnen andere afdelingen binnen ORTEC, gaat dit winst op leveren op de volgende manieren:

- Een nieuwe manier van datavisualisatie die in een aantal bestaande, maar ook toekomstige producten geïmplementeerd kan worden.
- Het standaardiseren van ORTEC stijl/uiterlijk in de datavisualisaties.
- Minder ontwikkelingskosten voor toekomstige implementatie van datavisualisatie door het hergebruiken van de gebouwde library.
- Het makkelijker realiseren van nieuwe visualisaties door functionaliteiten in de library te hergebruiken.
- Het makkelijker verhogen van de kwaliteit van meerdere projecten, door de gemaakte library te integreren in bestaande/nieuwe producten.

3.4.3 Testdocument

Na het ontwikkelen van de library, wordt er een testdocument opgesteld. Er is namelijk behoefte aan bewijsvoering van de werking van de library. Dit kan worden aangetoond door de werking van library te testen d.m.v. een implementatietest in een bestaand product en dit vast te leggen in een testdocument.

Tijdens de ontwikkeling van de library worden er ook Unit Tests geschreven. Alle testresultaten worden vastgelegd in het testdocument.

4 Aanpak

De afstudeerperiode wordt in een aantal fases doorlopen. 4.1 bevat de planning van deze fases en de toelichting hierop. In 4.2 t/m 4.5 wordt er toegelicht hoe deze fases individueel worden ingevuld.

Allereerst heb ik ingeschat dat er 3 belangrijke processen in de afstudeerperiode zijn die leiden tot een product:

1. opstellen van requirements;
2. ontwikkelen van de library;
3. testen van de ontwikkelde library.

Hier wordt al duidelijk dat elk proces pas kan worden gestart als de vorige is afgerond. Zo kan de library pas ontwikkeld worden als de requirements boven tafel zijn en de library kan pas getest worden als die ontwikkeld is. Dit is dus ook de grootste reden waarom er is gekozen om in fases te werken.

4.1 Globale planning/fasering

Om te kunnen te bepalen hoe de indeling van het afstudeertraject er globaal uit gaat zien, moet er eerst duidelijk worden welke fases er zijn en hoeveel tijd er beschikbaar is. Het afstudeertraject duurt 17 werkweken, dus: 17*5 = 85 werkdagen.

Allereerst moet er een aanpak bepaald worden voor hoe ik deze opdracht wil voltooien. Hier is de eerste week voor uitgetrokken. Het gaat hier om proces bepalen, vastleggen en inschatten wat de beste opties zijn in de gegeven context van de opdracht.

Vervolgens heb ik ingeschat dat ik het grootste gedeelte van de afstudeerperiode bezig ben aan de ontwikkeling van de library. Er is gekozen om hier ongeveer de helft van de afstudeerperiode (ongeveer 45 dagen) aan te werken, omdat dit het belangrijkste onderdeel is van de afstudeerperiode.

Het proces van documenteren van requirements is een belangrijk proces. Er is namelijk nog niet aan het begin van de afstudeerperiode bepaald wat de library voor functionaliteit moet bevatten. Het is belangrijk dat er in ieder geval een goede basis aan requirements gelegd wordt, voordat de ontwikkeling van de library kan beginnen.

Daarnaast is het ook een wens van ORTEC Sports dat de behoeften en eisen worden vastgelegd in een apart document, zodat er overeenstemming wordt bereikt m.b.t. de functionaliteit van de library. Er is dan ook 15 dagen ingeschat voor dit gehele proces.

Ook is duidelijk geworden dat er behoefte is aan bewijsvoering van de werking van de library in een of meerdere van de bestaande producten. Daarom heb ik besloten om, na de ontwikkeling van de library, de werking van de library te testen d.m.v. een implementatietest. Hier is dan ook 2 weken, of 10 dagen, voor ingeschat.

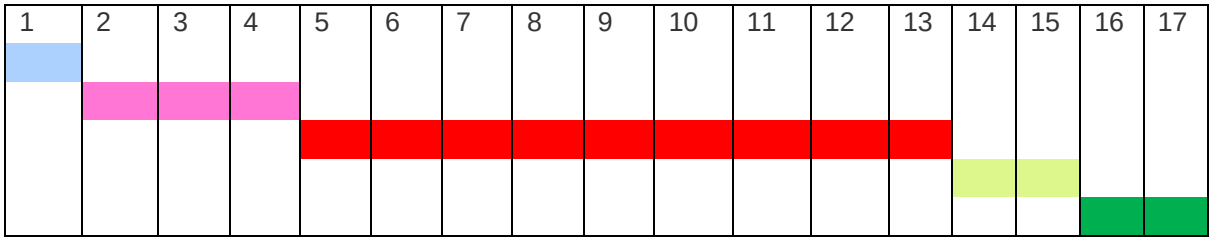
Allerlaatst heb ik mij laten adviseren door collega's dat het belangrijk is om een buffer aan het einde van de afstudeerperiode in te plannen, die zich volledig leent voor het afmaken van het afstudeerverslag. Hier zijn dan ook de laatste 2 weken voor ingeschat.

In tabel 3 en 4 is een overzicht te zien van de bepaalde fases en de daarbij behorende tijdsinschatting.

Tabel 3: Fases

Fase	Resultaat	Duur
Aanpak bepalen	Plan van aanpak	**5 dagen**
Requirements vastleggen	Requirements document	**15 dagen**
Ontwikkeling van library	Library	**45 dagen**
Testen	Testdocument	**10 dagen**
Afmaken afstudeerverslag	Afstudeerverslag	**10 dagen**

Tabel 4: Globale planning



4.2 Plan van aanpak

De eerste week van de afstudeerperiode wordt de aanpak gedefinieerd, die gehanteerd wordt tijdens de rest van de afstudeerperiode. Dit is een belangrijke eerste stap, omdat er dan een duidelijk proces ontstaat wat gevolgd kan worden. Deze aanpak wordt vastgelegd in het plan van aanpak. Dit onderdelen van het plan van aanpak worden verwerkt in het afstudeerverslag.

4.3 Requirements vastleggen

Allereerst worden er gesprekken gevoerd met de stakeholders. Dit zijn collega's van ORTEC Sports, zoals beschreven in hoofdstuk 2.4. Vanuit hier wordt verder bepaald wat de eisen en behoeften zijn. Deze eisen en behoeften worden vertaald naar requirements.

Vervolgens worden deze requirements geanalyseerd. Hier wordt bepaald in welke mate de requirements nog onduidelijk, incompleet of tegenstrijdig zijn. Deze opgestelde requirements worden terug gecommuniceerd naar de stakeholders voor controle. Na eventuele correctie worden de requirements gedocumenteerd en omgezet naar taken die uitgevoerd gaan worden tijdens het implementatieproces. Deze taken worden vastgelegd op een Kanban board, wat gebruikt wordt om de taken bij te houden.

Om de technische haalbaarheid te bepalen, worden de taken geprioriteerd met MoSCoW in overleg met de stakeholders. Dit geeft een duidelijk overzicht op welke taken noodzakelijk zijn om te volbrengen tijdens de ontwikkeling van de library.

4.4 Ontwikkeling van library

Nadat de taken zijn vastgelegd op het Kanban board, kan er een begin gemaakt worden aan de volgende fase: de ontwikkeling van de library.

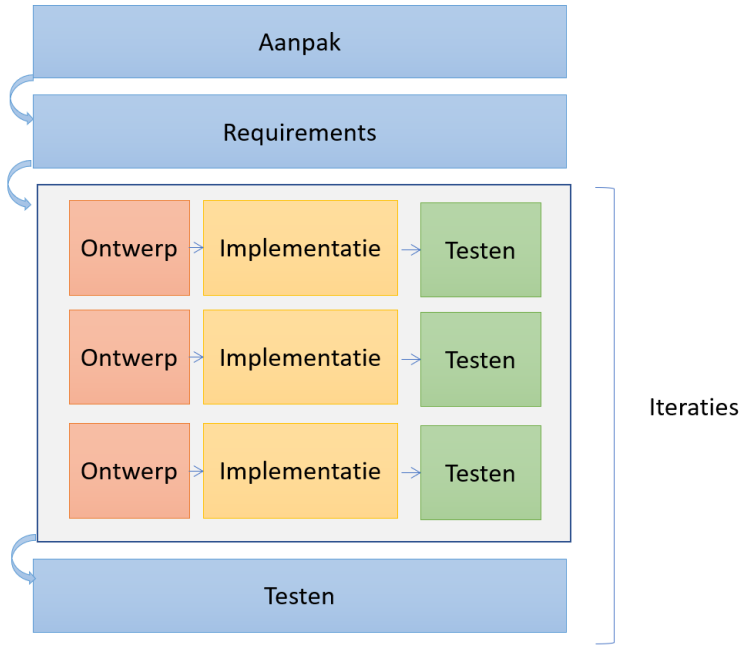
4.4.1 Methode

Tijdens de ontwikkeling van library wordt er Agile gewerkt. De grootste reden hiervoor is dat ik op deze manier in iteraties kan werken, waar ik aan het einde van elke iteratie een demo van de voortgang aan de stakeholders kan geven. Dit is een moment voor de stakeholders om hun input te geven, waarna deze eventueel kan worden meegenomen in een volgende iteratie. Op deze manier is het mogelijk om onderdelen, zoals de requirements, bij te stellen naar de eventuele veranderende vraag van de stakeholders.

Daarnaast is deze manier van werken een goede manier om het hoofdprobleem op te splitsen in sub problemen en op deze manier stap voor stap er doorheen te lopen. Er zijn namelijk een aantal taken die repetitief voor elk sub probleem moet worden uitgevoerd:

- Ontwerp
- Implementatie
- Testen (unit testing)

Figuur 3: Methode



Nadat de taken bekend zijn, wordt er op basis van de gemaakte prioriteit een keuze gemaakt welke taken die iteratie geïmplementeerd gaan worden. Er wordt een schatting gemaakt van welke functionaliteit er die iteratie wordt gemaakt. Vervolgens wordt deze functionaliteit ontworpen, geïmplementeerd en getest. Elke iteratie wordt gelijk ook de werking van de gebouwde functionaliteit getest. Aan het einde van een iteratie wordt de voortgang aan de stakeholders getoond, waarna er weer een volgende iteratie wordt ingepland. Op deze manier wordt door de iteraties heengelopen.

4.4.2 Ontwerp

Aan het begin van iedere iteratie worden er een technisch ontwerp gemaakt met eventueel een of meerdere UML-diagrammen van de te bouwen functionaliteit. Dit dient als beginpunt van de desbetreffende functionaliteit. Na de implementatie van deze functionaliteit, worden de UML-diagrammen eventueel bijgewerkt. Deze diagrammen dienen tevens als documentatie voor toekomstige ontwikkelaars.

4.4.3 Testen als onderdeel van iteratie

Iedere iteratie wordt er ook getest om de juistheid van een visualisatie aan te tonen. Dit heeft als doel om de juiste werking van een gemaakte component aan te tonen. Het gaat hier om unittesten, d.m.v. het testing framework Jasmine.

4.4.4 Reviews

Tijdens de afstudeerperiode worden alle vorderingen gereviewed, waaronder geschreven code, om de kwaliteit te waarborgen. Zo worden alle tussenproducten door de begeleider en/of overige collega's bekeken, waarna er feedback op gegeven wordt. Technisch worden moeilijke keuzes besproken met collega's, waarna er door mij een keuze wordt gemaakt op basis van deze gesprekken.

4.5 Implementatietest

Nadat de ontwikkeling klaar is, wordt de library d.m.v. een implementatietest in een van de bestaande producten getest.
Er is namelijk, zoals aangegeven in 3.4.3, behoefte aan bewijsvoering van de werking van de library in een bestaand product. Daarom is er gekozen om een test te houden waar een implementatie wordt gemaakt van de functionaliteiten van de library.

Het doel van deze test is om de juistheid van de library te testen. Dit wordt gedaan door een aantal acceptatiecriteria op te stellen op basis van de requirements. Vervolgens worden er testcases opgesteld en uitgevoerd o.b.v. deze acceptatiecriteria. Deze uitkomsten van de testen worden vastgelegd en op basis hiervan kan worden gesteld of deze testcase geslaagd is. De testcases en de uitkomsten hiervan worden vastgelegd in een testdocument.

5 Requirements

5.1 Initiële aanpak

Op basis van het afstudeerplan is er al een globaal idee van wat de functionaliteit van de library moet bevatten. Dit idee is alleen nog niet voldoende afgebakend, dus er is verdere analyse nodig.

Allereerst wordt er begonnen met het spreken van de stakeholders. Het doel is hier om boven water te krijgen wat de verwachtingen zijn van verschillende stakeholders.

Dit gebeurt d.m.v. gesprekken met de stakeholders, aangezien op deze manier er snel geïdentificeerd kan worden welke stakeholders welke verwachtingen hebben. Er is namelijk door mijn werkervaring op deze afdeling al geconstateerd dat ik met stakeholders te maken heb met verschillende achtergronden. Dit brengt het risico mee dat er verschillende eisen of verwachtingen uit de gesprekken naar voren zouden kunnen komen. Door in gesprek te gaan hierover met de stakeholders, kan dit probleem geconstateerd worden en vervolgens worden opgelost.

5.2 Tegenstrijdige verwachtingen stakeholders

Door het voeren van de gesprekken is duidelijk dat er onder de stakeholders tegenstrijdige verwachtingen zijn van het product. Kort samengevat zijn dit de twee verwachtingen:

- Een library waarin voltooide visualisaties worden aangeboden, zoals een piechart, voetbalveld of lijndiagram. Aan deze visualisaties kan input gegeven worden, waarna deze input gevisualiseerd wordt.

- Een meer generieke library waarin er verschillende elementen van een visualisatie, zoals lijnen en cirkels, makkelijker gebruikt kunnen worden. Met deze elementen kunnen uiteindelijk ook visualisaties zoals voetbalvelden of lijndiagrammen gemaakt worden. Kortgezegd: een tekenlibrary.

5.3 Stakeholder conflict

5.3.1 Probleem

Duidelijk is dat er twee compleet verschillende verwachtingen zijn van het te bouwen product. Dit is voor mij een probleem, omdat dit het risico vergroot dat er uiteindelijk een product wordt opgeleverd wat niet aan de eisen en verwachtingen voldoet. Ook moet er een beginpunt gecreëerd worden om aan de bouw van de library te beginnen. Dit kan alleen als de daadwerkelijke functionaliteit van de library goed is gespecificeerd.

5.3.2 Oplossing

Het is noodzakelijk om dit probleem uit de weg te helpen, zodat er de juiste requirements opgesteld kunnen worden. Om dit probleem op te lossen is er een klein rapport (zie Bijlage 1) geschreven, die de volgende onderwerpen bevat:

- de twee verwachtingen;
- voor- en nadelen van deze twee verwachtingen;
- de conflicten hiertussen;
- eigen advies over wat de beste oplossing zou zijn.

Door dit rapport te schrijven is er een overzicht van de situatie gecreëerd, voor zowel mijzelf als de stakeholders. Vervolgens heb ik dit rapport voorgelegd aan mijn leidinggevende, die in overleg met mij een uiteindelijk besluit heeft genomen. Het is dan ook uiteindelijk een middenweg geworden, namelijk een library waarin de tekenfunctionaliteit geëncapsuleerd is, waarmee er visualisaties gemaakt en vervolgens aangeboden worden.

5.4 Globale requirements

Nadat het stakeholder conflict is opgelost, moet er per visualisatie nagedacht worden over de eisen die hier aan verbonden zijn, en deze moet worden vastgelegd in het requirements document. Allerlaatst wordt dit weer gevalideerd door de stakeholders, waarna er aan de implementatie van het product begonnen kan worden.

De prioriteit ligt in ieder geval bij de veldvisualisaties, toegelicht in tabel 5. Deze requirements zijn terug te vinden in het requirements document (zie Bijlage 2), a.d.h.v. bijbehorende ID. Afhankelijk van de voortgang, wordt gekeken of er nog andere visualisaties gerealiseerd gaan worden.

Tabel 5: Veldvisualisaties

ID	Type visualisatie	Toelichting	Voorbeeld
5	Veldvisualisatie	Dient als basis voor de rest van de visualisatie.	Een heel of half voetbalveld.
17	Veld locaties	Actiepunten op een veldvisualisatie, gevisualiseerd met vormen en kleuren, die staan voor bepaalde acties.	Een schot op goal van buiten het strafschopgebied, gevisualiseerd op het corresponderende XY-coördinaat.
15	Veldlijnen	Actiepunten op een veldvisualisatie met een bepaalde richting.	Een voorzet die gegeven wordt vanuit een startpunt en een bepaalde richting op gaat.
10	Veld grid	Mogelijkheid om veld onder te verdelen in vakken, of ook wel een grid, en hier waardes in te tonen.	Gemiddeld aantal passes of schoten vanuit een bepaalde zone(vak).
7	Passmap	Passes die spelers onderling naar elkaar geven, gevisualiseerd met lijnen en bolletjes.	Passes in een wedstrijd van Ajax, waarin te zien is welke spelers er de meeste passes naar elkaar verstuurd hebben.
19	Action replay	Een herhaling van een bepaald evenement in een wedstrijd, gevisualiseerd o.b.v. datapunten.	Een goal van Ajax waar 7 spelers bij betrokken zijn. Elke actie in deze aanval wordt stap voor stap chronologisch herhaald.

6 Implementatie

6.1 Iteratie 1: Begin bouw veldvisualisaties

6.1.1 Inschatting te implementeren functionaliteit

In samenspraak met de opdrachtgever is tijdens de requirementsfase besloten dat de veldvisualisaties een Must Have zijn (zie Bijlage 2), dus er is dan ook gekozen om daar mee te beginnen tijdens de eerste iteratie.

Voor de eerste iteratie is er 3 weken ingepland, dus 15 beschikbare werkdagen.

Allereerst wordt er een standaard veldvisualisatie gemaakt, omdat dit de basis is voor bijna alle andere visualisaties. Vervolgens is er gekozen de volgende visualisaties te maken tijdens de eerste iteratie:

Tabel 6: Inschatting iteratie 1

Requirement ID	Werkzaamheden	Schatting dagen
	Ontwerp	2
5	Heel en half veld visualisatie implementatie	4
17	Veld locaties	4
15	Veldlijnen	4
	Testen	1/2

Dit brengt het totaal op 15 à 16 dagen voor de eerste iteratie. Hierin zitten ook overige werkzaamheden die bij een eerste iteratie komen kijken zoals: opzetten van programmeeromgeving, versiebeheer, taken neerzetten op het Kanban board etc.

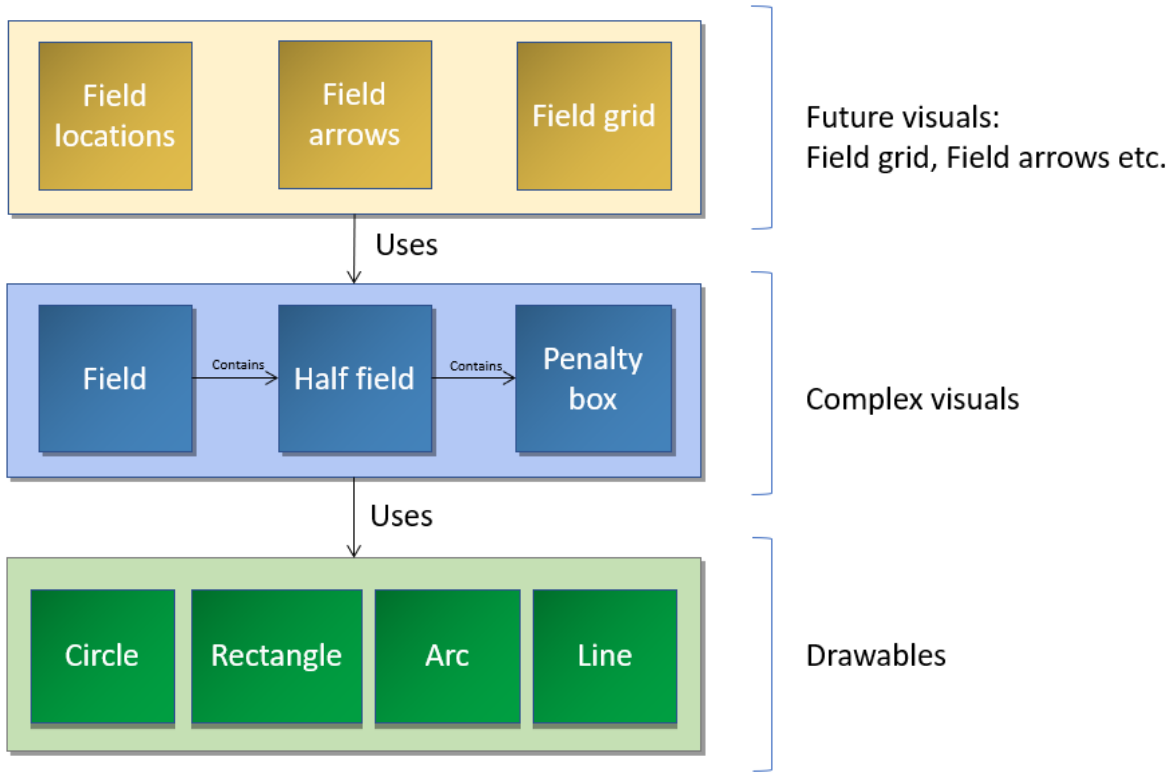
6.1.2 Probleem veldvisualisaties

Aan het begin van de eerste iteratie wordt er met behulp van de D3.js library een simpele veldvisualisatie gemaakt in een Angular component. Deze visualisatie wordt opgebouwd door statische TypeScript code die uiteindelijk wordt aangeroepen als het desbetreffende component wordt aangemaakt. Dit is een oplossing die werkt, alleen het brengt wel een aantal problemen met zich mee:

- Statische D3 code: moeilijk aan te passen, dus niet goed onderhoudbaar.
- Duplicaat code: voor een half veld moet nog steeds de helft van dezelfde code geschreven worden als bij een heel veld.
- Elementen in een veldvisualisatie, zoals lijnen en cirkels, zijn niet los herbruikbaar. Voor elk element moet dezelfde D3 code geschreven worden om deze elementen te kunnen tekenen.

Om deze problemen op te lossen kies ik er voor om een andere abstractie te kiezen. Ik kies er voor om een aparte laag voor alle basis-tekenfunctionaliteit te maken en hiermee alle visualisaties op te bouwen.

Figuur 4: Opbouw visualisaties

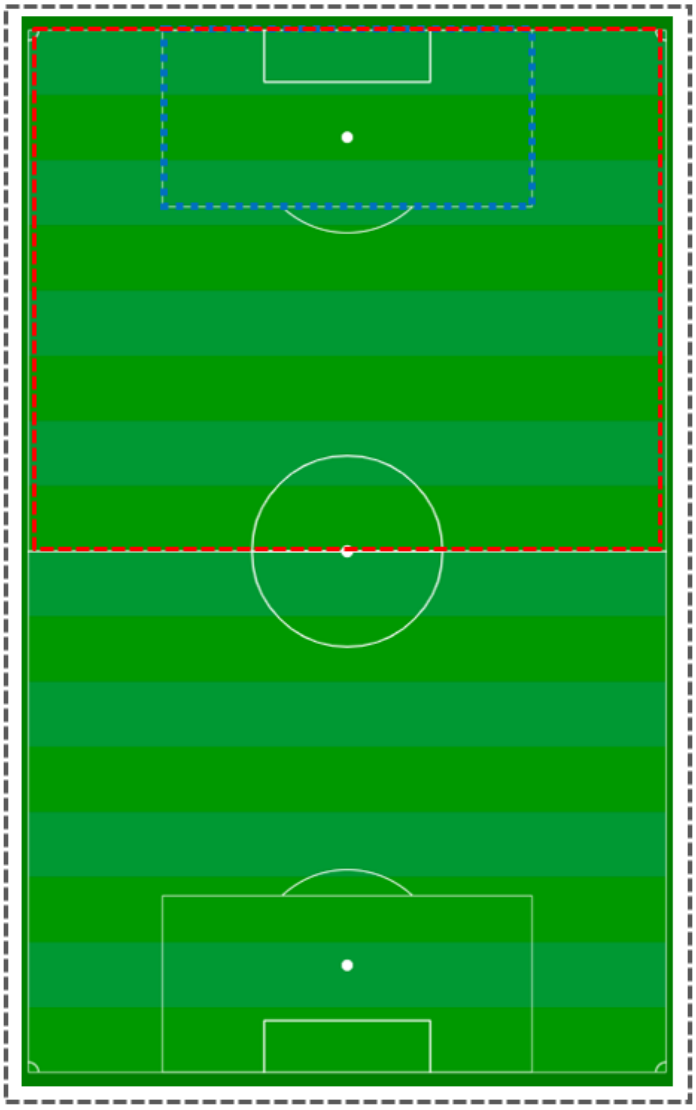


Door een andere abstractie te kiezen voor de tekenfunctionaliteit ontstaat een gelaagde architectuur waarin de functionaliteit van het tekenen van vormen geëncapsuleerd is. In figuur 4 is deze architectuur visueel uitgebeeld. Als onderste laag zijn daar de basisvormen: de drawables, zoals een cirkel of een driehoek. Met deze basisvormen kunnen er complexe visualisaties gemaakt worden zoals een strafschopgebied, die zelf ook weer herbruikbaar is in een half veld. Toekomstige visualisaties kunnen door deze structuur ook weer de complexe visualisaties uitbreiden, zoals bijvoorbeeld een field grid (Requirement ID: 10).

Deze keuze brengt een aantal voordelen met zich mee:

- Elk element in een visualisatie is een zelfstandig object, dus ook simpel manipuleerbaar. Operaties zoals het dynamisch weghalen of opnieuw tekenen van een element is hierdoor mogelijk.
- Elke laag is zelf uitbreidbaar, zo kan er bijvoorbeeld makkelijk een nieuwe vorm toegevoegd en gebruikt worden.
- Toekomstige visualisaties zijn eenvoudiger te bouwen, doordat de kernfunctionaliteit van een element tekenen geëncapsuleerd is. Dit zorgt voor een hogere uitbreidbaarheid, doordat de tekenfunctionaliteit voor iedere toekomstige visualisatie te gebruiken is.
- Elk onderdeel van een visualisatie is op deze manier los opgebouwd en dus herbruikbaar in andere visualisatie. Een penaltygebied wordt opgebouwd als een los object, die herbuikbaar is in bijvoorbeeld een half veld.

Figuur 5: Voetbalveld opgebouwd uit losse objecten



Deze keuze lost de volgende problemen op:

- Er is geen statische D3.js code, maar geïsoleerde objecten: visualisaties zijn hierdoor beter aanpasbaar, dus beter onderhoudbaar.
- Geen duplicaat code voor bijvoorbeeld een heel of half veld: een heel veld creëert nu twee objecten van halve velden i.p.v. zelf twee keer de code van een half veld te hoeven schrijven.

6.1.3 Probleem schalen van coördinaten

Tijdens de bouw van de veldvisualisaties wordt vanuit de opdrachtgever duidelijk dat de datapunten die op het veld worden getoond, opgeslagen worden in een 100 bij 100 coördinatensysteem. Dit levert voor een voetbalveld een aantal problemen op:

- Een voetbalveld heeft niet een 1 op 1 ratio van 100 bij 100, maar heeft een lengte van 120 en een breedte van 75. De lengte is altijd 1,6 keer groter dan de breedte.
- Coördinaten en elementen schalen niet standaard op de juiste manier mee. Hierdoor worden elementen met verkeerde afmetingen op verkeerde plekken getekend. Een element met een coördinaat van [50,50] komt daardoor niet in het midden te staan op een veld van 120 bij 75, terwijl dit wel moet.

Aangezien de gehele library gebouwd is met D3.js, zou het natuurlijk ideaal zijn als hier al een oplossing voor is ingebouwd. Als die oplossing er nog niet is, dan kan die altijd nog zelf worden gebouwd. Na verder onderzoek zijn er deze opties gevonden om het schaalprobleem op te lossen:

Tabel 7: Voor- en nadelen schalen van coördinaten

Oplossing	Voordelen	Nadelen
ViewBox [4]	• Introduceert nieuw coördinatensysteem binnen SVG.	• Rekt elementen uit om aan juiste waardes te voldoen • Moet alsnog een eigen schaling gebouwd worden om uitrekken te voorkomen.
D3.js linear scaling[5]	• Input van dataset min/max (100 bij 100) en afmetingen van een SVG en mapt dit naar elkaar. • Elementen worden op juiste plek getekend. • Elementen worden niet uitgerekt.	• Afmetingen van elementen worden geconverteerd aan de hand van de hoogte/breedte ratio, bijvoorbeeld: element van 2 bij 2 is dus geen vierkant.
Zelf schaling bouwen	• Meer controle over schaling.	• Veel onderhoud: complexe objecten. • Gevoelig voor afrondingsfouten. • Meerderheid van operaties zijn al beschikbaar in D3.

[4] https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/viewBox

[5] https://d3indepth.com/scales/

In tabel 7 zijn de voor- en nadelen beschreven van mogelijke oplossingen voor het schaalprobleem. Van zowel de Viewbox als de D3.js linear scaling oplossing, zijn er implementaties gemaakt waar deze voor- en nadelen uit kwamen.

Vervolgens wordt er met collega's gesproken over wat het inhoudt om zelf de schaling in te bouwen. Hieruit komt vooral naar voren dat dit een onnodig complexe oplossing is en er mogelijk problemen kunnen ontstaan met het afronden van de berekeningen. Daarnaast is deze oplossing ook deels opnieuw het wiel uitvinden, want D3.js heeft al veel van dit soort operaties verwerkt in bijvoorbeeld linear scaling.

De uiteindelijke keuze valt op D3.js linear scaling:

- Creëert een nieuw coördinatensysteem en mapt dit naar de corresponderende hoogte en breedte.
- Elementen worden op de juiste plek getekend en niet uitgerekt zoals bij de ViewBox.
- Linear scaling verwacht een input van min/max dataset. Dit maakt de input variabel en dus ook te gebruiken bij bijvoorbeeld een half veld of penalty box, waar er andere coördinatensystemen gelden.

Uiteindelijk zijn deze voordelen toereikend genoeg om het probleem van schalen op te lossen.

6.1.4 Evaluatie iteratie 1

Aan het begin van iteratie 1 had ik ingeschat dat ik verschillende veldvisualisatie af zou krijgen. Uiteindelijk heb ik alleen het veld, halve veld en het strafschopgebied afgekregen. Dit komt vooral omdat ik de focus heb gelegd op de kern van de library: de tekenfunctionaliteit.

Het is namelijk essentieel voor de library dat de implementatie tekenfunctionaliteit op zichzelf goed kan werken. Deze implementatie zorgt er namelijk ook voor dat er herbruikbare objecten gecreëerd kunnen worden, zoals bijvoorbeeld een strafschopgebied of een half veld. Toekomstige visualisaties, zoals bijvoorbeeld veldlocaties, zijn nu veel makkelijker te bouwen en onderhouden dan voorheen. Hiermee worden belangrijke softwarekwaliteitskenmerken zoals onderhoud- en uitbreidbaarheid gewaarborgd.

Hoewel de eerste iteratie geen concrete (visuele) resultaten heeft opgeleverd, is naar mijn mening niet erg. Er is namelijk een betere basis gelegd voor toekomstige visualisaties. De verwachting is dan ook dat er de volgende iteratie grotere stappen gemaakt kunnen worden met het realiseren van nieuwe visualisaties.

6.1.5 Demo iteratie 1

Zoals beschreven in de aanpak, heb ik gekozen om na iedere iteratie een demo te geven aan de stakeholders, zodat zij hun feedback kunnen geven op de gebouwde functionaliteit. Er is alleen door mij gekozen om dat na deze iteratie niet te doen.

Deze iteratie bestond vooral uit technische oplossingen, namelijk het tekengedeelte van de library. Deze technische oplossingen zijn tijdens de implementatie al uitvoerig besproken en gereviewed met de betrokken collega's, die ook stakeholder zijn. Het heeft om die redenen volgens mij ook geen zin om dat nog een keer te bespreken met diezelfde stakeholders.

Daarnaast is het zo dat er deze iteratie nog niks visueels gerealiseerd is. Er valt dus ook niks visueels te tonen tijdens een demo. De gemaakte voortgang is voor het merendeel van de stakeholders dus niet relevant. De verwachting is dat dit tijdens iteratie 2 wel gaat gebeuren.

6.2 Iteratie 2: Complexe visualisaties

6.2.1 Inschatting te implementeren functionaliteit

De vorige iteratie is er gewerkt aan de tekenfunctionaliteit van de library en hiermee zijn veldvisualisaties opgebouwd, zoals een heel of half voetbalveld. Deze iteratie worden meer complexe visualisaties gebouwd bovenop deze veldvisualisaties. Het plan is om deze iteratie de volgende visualisaties te maken:

Tabel 8: Inschatting iteratie 2

Requirement ID	Werkzaamheden	Schatting dagen
	Ontwerp	2
10	Veld grid	4
17	Veld locaties	4
15	Veldlijnen	4
	Testen	1/2

6.2.2 Verandering van raamwerk

Na voltooiing van de eerste iteratie en tijdens de tweede iteratie wordt duidelijk dat de functionaliteiten die het Angular raamwerk biedt nauwelijks gebruikt worden. Het is namelijk zo dat ik de visualisaties bouw met de D3.js library. Deze library genereert SVG's vanuit D3-code geschreven in Javascript, in mijn geval Typescript.

Angular daarentegen versimpelt en verbetert de interactie tussen HTML, CSS en Javascript. Daarnaast leent het zich makkelijk om webapplicaties of SPA's (Single Page Applications) te realiseren. Functionaliteiten zoals routing zijn bijvoorbeeld ingebouwd in het raamwerk. Dit zijn allemaal functionaliteiten die ik niet nodig heb om de visualisaties te creëren.

Vervolgens worden de voor- en nadelen afgewogen, te zien in tabel 9, wat voor gevolgen het zou hebben wanneer er afgestapt wordt van het Angular raamwerk en de library in alleen TypeScript wordt geschreven:

Tabel 9: Voor- en nadelen raamwerk keuze

Raamwerk	Voordelen	Nadelen
Angular	• Verbetert en versimpelt interactie tussen HTML, CSS en JavaScript (TypeScript). • Leent zich makkelijk om webapplicaties of SPA's (Single Page Applications) te realiseren.	• Brengt overbodige configuratie met zich mee. • Brengt overbodige functionaliteit met zich mee. • Extra compileerstap naar Web Components, om herbruikbaar te zijn in elk project.
TypeScript only	• Compileert naar Javascript en is herbruikbaar in ieder project, want draait in iedere browser. • Brengt geen overbodige functionaliteit mee.	• Functionaliteit die het Angular raamwerk aanbiedt, moet zelf worden gebouwd, indien nodig.

Waarom is Angular dan nog nodig? Het brengt namelijk heel veel voordelen mee als in het geval van dat er daadwerkelijk een webapplicatie gebouwd wordt. In mijn geval tellen die voordelen allemaal niet. Sterker nog; het brengt alleen maar nadelen mee, zoals de overbodige configuratie en functionaliteit.

Als de library alleen in TypeScript wordt geschreven, wat ik eigenlijk tot nu toe ook heb gedaan, dan worden alle nadelen geëlimineerd die worden ondervonden door gebruik te maken van Angular.

Daarnaast brengt het ook een groot voordeel mee: herbruikbaar in ieder project, want draait in iedere browser. Oorspronkelijk was het de bedoeling om deze compatibiliteit te bereiken door alle visualisaties te compileren van Angular Components[6] naar Web Components. Deze stap is dus overbodig en wordt ook geëlimineerd.

Deze bevindingen worden voorgelegd aan mijn leidinggevende, waarna hij ook vindt dat het Angular framework overbodig is voor de library die ik aan het bouwen ben.

[6] https://angular.io/guide/elements

Vervolgens heb ik besloten om door te gaan zonder Angular en de library volledig in Typescript te schrijven. Alle visualisaties worden in Typescript classes i.p.v. Web Components[7] aangeboden aan de consumer van de library.

6.2.3 Factory

Tijdens het ontwerp van de veld locaties-visualisatie worden een aantal stappen duidelijk, die moeten gebeuren tijdens de opbouw van de visualisatie:

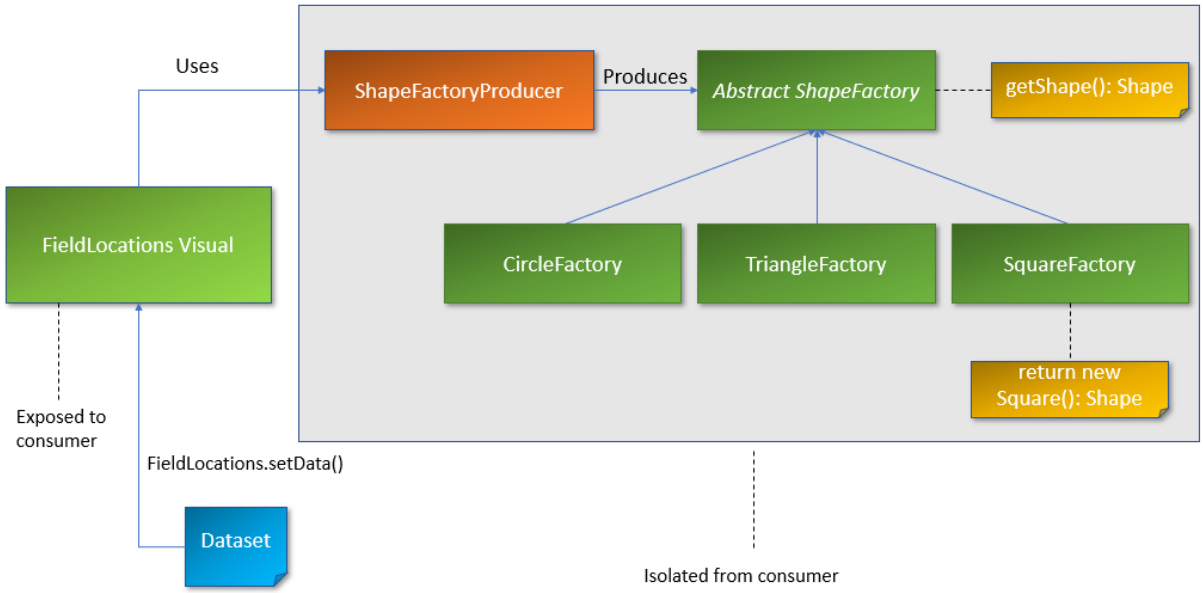
1. Een dataset met XY-coördinaten moet als input dienen.
2. Per punt moet er geconfigureerd zijn om wat voor vorm (cirkel, driehoek) en kleur het gaat.
3. Voor ieder datapunt in de dataset, moet er een punt op het voetbalveld getekend worden volgens de juiste configuratie.

Een voorbeeld hiervan kan zijn: op coördinaat [50,20] moet er een blauwe driehoek getekend worden, want dit illustreert een schot op goal van het blauwe team.

Om deze stappen te voltooien, zijn er een aantal verantwoordelijkheden binnen de library:

- het ontvangen van de dataset;
- het bepalen welk object gecreëerd moet worden;
- het creëren van het object met de juiste waardes;
- het daadwerkelijk visualiseren of tekenen van de objecten op het veld

Figuur 6: Opbouw ShapeFactory



[7] https://developer.mozilla.org/en-US/docs/Web/Web_Components

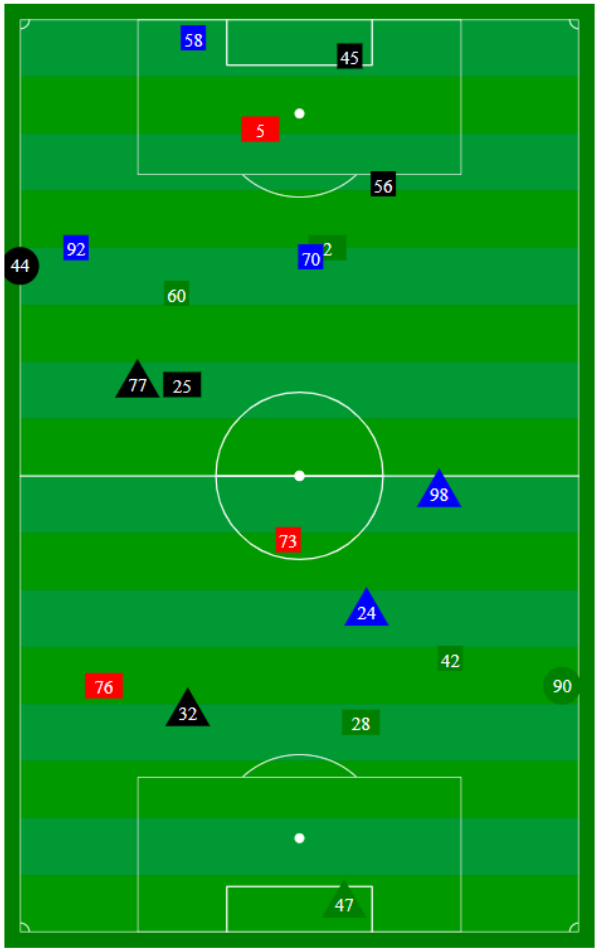
Om deze verantwoordelijkheden op de juiste manier te verdelen, is een abstract factory pattern de ideale oplossing. In figuur 6 is een visualisatie van hoe dit design pattern wordt geïmplementeerd. Duidelijk is dat er de consumer van de library alleen maar de veldvisualisaties, in dit geval de FieldLocations visualisatie, kan gebruiken. De rest van de logica is geisoleerd.

Tabel 10: Verdeling van verantwoordelijkheden

Object	**Verantwoordelijkheid**
FieldLocations Visual	Ontvangen van dataset.
ShapeFactoryProducer	Bepaalt welk object gecreëerd moet worden en creëert hiervoor de juiste factory.
Abstract ShapeFactory	Creeert het juiste Shape object met de juiste waardes.
Shape object	Tekent zichzelf in de FieldLocations Visual.

Zoals in tabel 10 is te zien, is er nu een duidelijke verdeling van verantwoordelijkheden gecreëerd door het abstract factory pattern toe te passen.

Figuur 7: Shapes geproduceerd door factory



In figuur 7 is het uiteindelijke resultaat te zien van de factory. Elke shape wordt via de factory geproduceerd en vervolgens getekend in de veldlocaties-visualisatie.

6.2.4 Evaluatie iteratie 2

Aan het begin van iteratie 2 heb ik ingeschat om de drie visualisatie te implementeren tijdens deze iteratie:

- Veld locaties (Requirement ID: 17)
- Veld lijnen (Requirement ID: 15)
- Veld grid (Requirement ID: 5)

Uiteindelijk zijn deze visualisaties, op de veld grid na, allemaal af gekomen. Er is deze iteratie veel tijd gaan zitten in de verandering van architectuur van Angular naar Typescript only. Dat is dan ook de voornaamste reden waarom niet alle vooraf ingeschatte functionaliteit geïmplementeerd is.

Naar mijn mening is het niet erg dat alles niet af is, want het blijft een inschatting. Het is vooral belangrijk dat softwarekwaliteitskenmerken zoals onderhoudbaarheid gewaarborgd worden. Dit wordt juist gewaarborgd door noodzakelijke veranderingen zoals het veranderen van het raamwerk naar TypesScript only. Zo is de onderhoudbaarheid door deze verandering vergroot, doordat onnodige Angular-code vermeden wordt, en er minder configuratie is. Dat er dan een visualisatie meer of minder niet af komt tijdens een iteratie, is dan ook niet erg.

6.2.5 Demo iteratie 2

Tijdens het presenteren van de voortgang werd duidelijk dat het de goede kant op gaat. Ook kwamen zones op het veld ter sprake. Zo kwam de wens naar voren om een zone op het veld dynamisch te selecteren en hier de datapunten te filteren. Dit moest in principe op elke visualisatie mogelijk zijn. De afspraak is gemaakt om hier prioriteit aan te geven tijdens iteratie 3 en daarna pas de focus te leggen op nieuwe visualisaties.

6.3 Iteratie 3: Dynamisch selecteren en nieuwe visualisaties

6.3.1 Inschatting te implementeren functionaliteit

De vorige iteratie is er gewerkt aan het ontwikkelen van visualisaties en de migratie van Angular naar TypeScript only. Uit de demo van iteratie 2 bleek dat er behoefte was vanuit de stakeholders om zones op een veld te kunnen selecteren. Het plan is om de volgende functionaliteit te implementeren in iteratie 3:

Tabel 11: Inschatting iteratie 3

Requirement ID	Werkzaamheden	Schatting dagen
	Ontwerp	1
	Dynamische select	4
10	Veld grid	1
19	Action replay	5
7	Pass map	3
	Testen	1

6.3.2 Dynamisch selecteren van zones op een veld

6.3.2.1 Analyseren en opsplitsen van de problemen

Tijdens het ontwerpen van het dynamisch selecteren van zones op een veld worden een aantal stappen duidelijk die moeten gebeuren tijdens de bouw hiervan:

1. Een gebruiker moet met zijn muis een selectie kunnen maken.
2. Er moet een rechthoek verschijnen die het geselecteerde gebied visualiseert.
3. Nadat de gebruiker zijn muis los laat, moeten de datapunten die niet binnen dat gebied vallen, uit het veld gefilterd worden.
4. Wanneer een gebruiker klikt wanneer hij een gebied heeft geselecteerd, dan moet het geselecteerde gebied verdwijnen en moet alle data weer getoond worden.

Vervolgens zijn de volgende verantwoordelijkheden geconstateerd die deze functionaliteiten met zich mee brengen:

- Het opvangen en uitlezen van de mouse events in de browser.
- Het tekenen en weer weghalen van een rechthoek o.b.v. deze mouse events.
- Het filteren van de data in dat geselecteerde gebied.

6.3.2.2 Hoe communicatie in de library van mouse events afhandelen?

Allereerst moeten de muis events opgevangen worden en op basis hiervan moet een rechthoek getekend worden die het geselecteerde gebied visualiseert. Door de event listeners van de D3.js library[8] te gebruiken, wordt dit ingebouwd.

Hoe worden de geselecteerde coördinaten aan de juiste objecten doorgegeven, zodat zaken zoals filtering afgehandeld kunnen worden? Eigenlijk moet er op basis van een event wat getriggerd wordt door de gebruiker, een stuk code worden uitgevoerd. Hiervoor zijn een aantal messaging patterns, maar de twee van de meest gebruikte voor dit probleem zijn:

- Observer Pattern
- Publish-Subscribe Pattern

In tabel 12 zijn de belangrijkste voor- en nadelen van deze design patterns opgesteld.

[8] https://www.dashingd3js.com/lessons/d3-and-dom-events

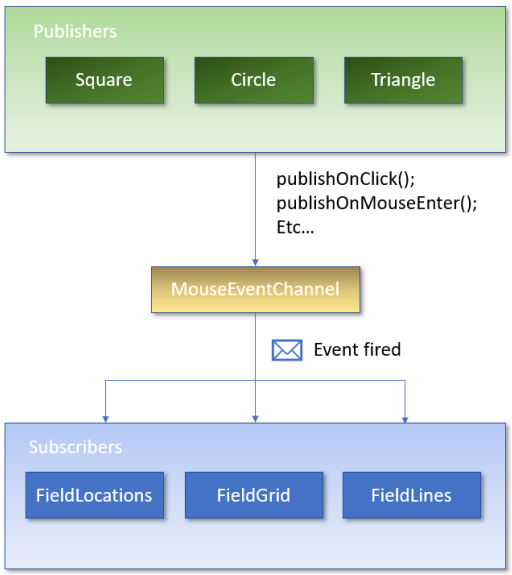
Tabel 12: Voor- en nadelen Observer en Pub-Sub pattern

Pattern	Voordelen	Nadelen
Observer[9]	• Mogelijkheid om data naar verschillende objecten tegelijk te sturen. • Ontvangers van een event kunnen op elk moment toegevoegd of verwijderd worden: schaalbaarheid. • Minimale afhankelijkheid tussen verzender en ontvanger.	• Observer is verplicht het gedrag, in de vorm van een interface, te implementeren. • Een verandering in de geïmplementeerde interface, moet voor alle observers geïmplementeerd worden: veel onderhoud
Pub-sub[10]	• Losse koppeling tussen publishers en subscribers: ze weten niet van elkaars bestaan. Een subscriber luistert naar een type bericht, niet naar de publisher van dat bericht. • Subscriber heeft de keuze naar welk type bericht hij wil luisteren en is niet verplicht om onnodig gedrag van een interface te implementeren. • Middelste laag is uitbreidbaar, zonder dat subscribers of publishers iets moeten veranderen: hoge uitbreidbaarheid en losse koppeling. • Hoge testbaarheid: makkelijk om te achterhalen of een subscriber het juiste type bericht krijgt.	• Losse koppeling is ook een nadeel: publishers hebben geen weet van of het bericht succesvol is aangekomen bij de subscriber. • Onmogelijk voor de publisher om te achterhalen of het gegenereerde bericht ook bij de juiste subscriber is aangekomen.

[9] https://medium.com/datadriveninvestor/design-patterns-a-quick-guide-to-observer-pattern-d0622145d6c2

[10] https://aws.amazon.com/pub-sub-messaging/

Figuur 8: Publish-Subscribe pattern



De keuze is uiteindelijk gevallen, zoals te zien in figuur 8, op het publish subscribe pattern om de volgende redenen:

1. Minder onderhoud aan de ontvangers van het bericht: zij kiezen zelf welk type bericht zij ontvangen.
2. Centralisatie van muis events: alle communicatie van muis events kan in 1 object afgehandeld worden. Toekomstige implementaties, zoals bijvoorbeeld hovers over veldobjecten, kunnen via dit centrale object afgehandeld worden en gedistribueerd naar de juiste objecten.
3. Losse koppeling: alle drawables (publishers), zoals cirkels of rechthoeken, hoeven geen weet te hebben van alle ontvangers (subscribers).
4. Onnodige complexiteit wordt door losse koppeling vermeden: een veelgebruikte operatie zoals het dynamisch verwijderen en opnieuw creëren van veldobjecten, heeft geen gevolgen voor de subscribers. In het geval van een observer pattern zou dit gevolgen hebben voor de ontvangers van het bericht, omdat zij telkens weer moeten registreren bij een nieuwe instantie van zo'n veldobject.

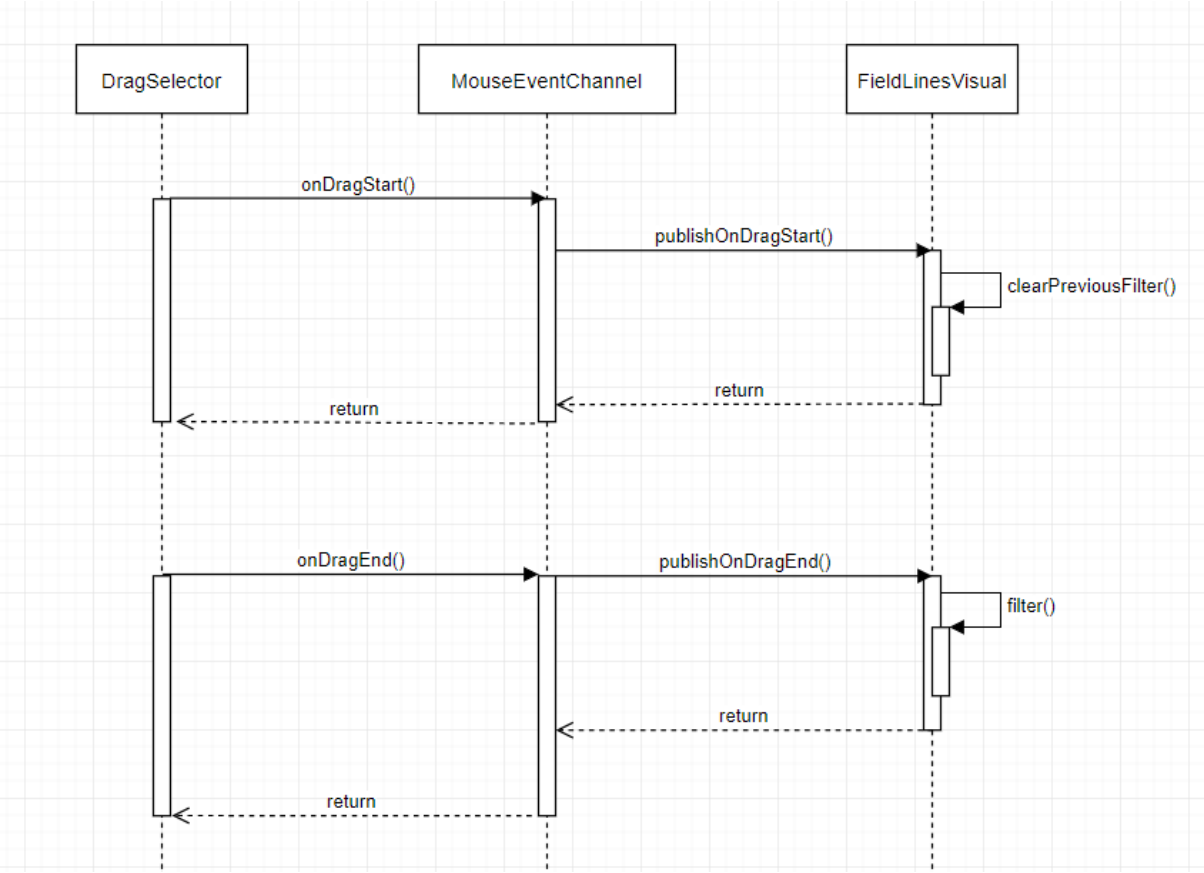
Door het toepassen van dit pattern ontstaat er voor de functionaliteit van het dynamisch selecteren de volgende onderverdeling verantwoordelijkheden:

Tabel 13: Verdeling van verantwoordelijkheden

Object	Verantwoordelijkheden
DraggableRectangle (publisher)	• Opvangen en uitlezen van mouse events. • Tekenen van rectangle op basis van mouse events. • Publiceren van mouse events naar MouseEventChannel.
MouseEventChannel	• Ontvangen van door publisher gegenereerde mouse events. • Subscribers op de hoogte stellen van deze events.
FieldVisual (subscriber)	• Ontvangen van message van MouseEventChannel en op basis hiervan filtering toepassen.

Hierdoor ontstaat een meer gelaagde en losgekoppelde communicatie tussen deze objecten. De FieldVisual is compleet losgekoppeld van alle functionaliteit m.b.t. het dynamisch selecteren van een zone op het veld. FieldVisual hoeft alleen maar te luisteren naar berichten naar keuze van MouseEventChannel, waarna hij zelf implementatie geeft aan de vervolgstappen.

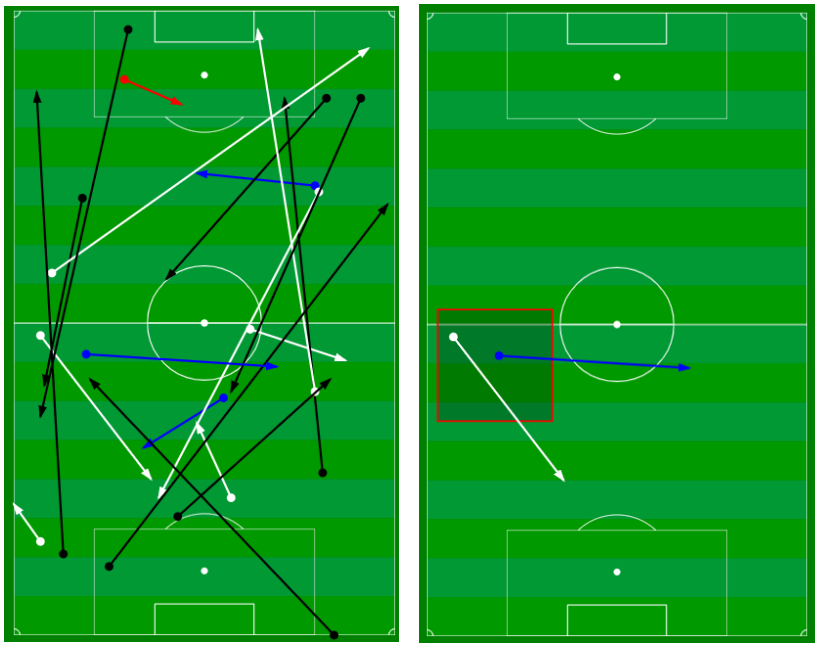
Figuur 9: publisher-subscriber pattern sequence



In figuur 9 is een illustratie te zien van hoe het publisher-subscriber pattern is geïmplementeerd in combinatie met de DragSelector in de library:

- DragSelector vangt events op en geeft dit door aan MouseEventsChannel.
- MouseEventChannel geeft dit door aan de subscribers van dit event.
- FieldLinesVisual is een subscriber en krijgt een notificatie van dit event, met bijbehorende coördinaten van het geselecteerde gebied, waarna er filtering in de data kan plaatsvinden.

Figuur 10: werking dynamisch selecteren zone



In figuur 10 is het uiteindelijke resultaat van het dynamisch selecteren te zien. Een gebruiker kan overal op het veld een zone ‘draggen’ met zijn muis, waarna datapunten in deze zone gefilterd worden.

6.3.3 Grid plot visualisatie

Uit de requirements kwam naar voren dat de Grid plot visualisatie moet dienen voor cases zoals een aggregatie van alle doelpogingen van een team tijdens een seizoen, in een bepaald gebied of vak op het veld.

Hier zijn de volgende eisen aan verbonden:

- Elk vak moet dynamisch configureerbaar zijn: er moet ingesteld kunnen worden waar het vak getekend wordt en hoe groot het vak moet zijn.
- Voor elk vak moet de achtergrondkleur instelbaar zijn.
- Voor elk vak moet er data ingesteld en getoond kunnen worden.

Vervolgens moet er worden onderzocht worden wat nou eigenlijk het moeilijkste onderdeel van deze visualisatie was. Er moet namelijk bepaald worden wat de nodige input is voor het creëren van de vakken. Het belangrijkste is dat dat de vakken op de juiste posities en met de juiste grootte getekend worden.

Om een vak op deze manier te tekenen, er van uitgaande dat alle elementen op een veld vanuit linksboven getekend worden, zijn vak eigenlijk maar 2 coördinaten nodig:

- Coördinaat linksboven van het vak.
- Coördinaat rechtsonder van het vak.

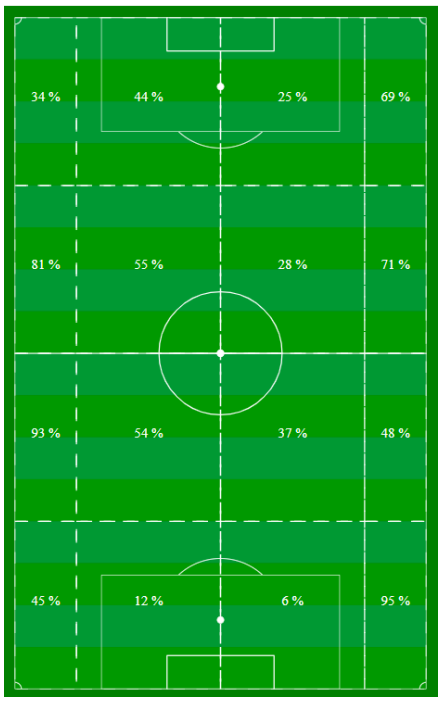
Met deze 2 coördinaten kan zowel de positie als de grootte van het vak berekend worden. Allerlaatst kan er per vak nog een tekst, achtergrond- en lijnkleur als input meegegeven worden. Dit is dan ook de gekozen input voor de Grid plot visualisatie.

Verder hoeft er voor deze visualisatie niet veel meer gebouwd te worden, aangezien de objecten die voor deze visualisatie nodig zijn, namelijk rechthoeken, al gebouwd zijn in iteratie 1 en kunnen worden hergebruikt.

6.3.3.1 Grid plot resultaat

In figuur 11 is het uiteindelijke resultaat te zien van de Grid plot visualisatie. De verdeling van de vakken is maar een voorbeeld van hoe dit gebruikt kan worden. Het veld kan namelijk op elke manier worden opgedeeld. Daarnaast zijn het willekeurige waardes die getoond worden in de vakken.

Figuur 11: Grid plot visualisatie



De verwachting is dat deze visualisatie vooral gebruikt kunnen worden voor situaties zoals aantal passes per zone. Het kan bijvoorbeeld ook gebruikt worden om de positie van aangekomen voorzetten te visualiseren. Doordat de grid dynamisch op te bouwen is, is het heel flexibel en kan er in principe allerlei soorten visualisaties mee gecreëerd worden

6.3.4 Action Replay visualisatie

Allerlaatst wordt er in iteratie 3 gewerkt aan de Action Replay visualisatie. Deze visualisatie heeft als doel een herhaling te tonen van een bepaalde actie of evenement in een wedstrijd, op basis van datapunten.

Tijdens het ontwerp en uit de requirements van deze visualisatie worden een aantal functionaliteiten duidelijk die de visualisatie moet bevatten:

1. Een lijst aan evenementen moet als input gegeven kunnen worden.
2. De Action Replay moet gepauzeerd kunnen worden.
3. De Action Replay moet hervat kunnen worden.
4. De Action Replay moet gestopt kunnen worden.
5. Er moet ingesteld kunnen worden hoe snel de herhaling wordt afgespeeld.

6.3.4.1 Het afspelen van de events

Het grootste gedeelte van deze visualisatie bestaat uit het afspelen van de events. Hoe dit op te lossen?

Allereerst moet de visualisatie de input omzetten naar objecten die getekend moeten worden. Dit is een minder complexe stap, omdat hiervoor de abstractie laag die gebouwd is in iteratie 1 en de daar bijhorende factories kunnen worden hergebruikt.

Wanneer deze objecten gecreëerd zijn, moeten ze stuk voor stuk op een bepaald tempo worden moeten worden getekend. Hiervoor gebruik ik de standaard interval timer die in JavaScript en TypeScript is ingebouwd. Met deze timer is namelijk de snelheid ook instelbaar, en dat is een van de eisen aan deze visualisatie.

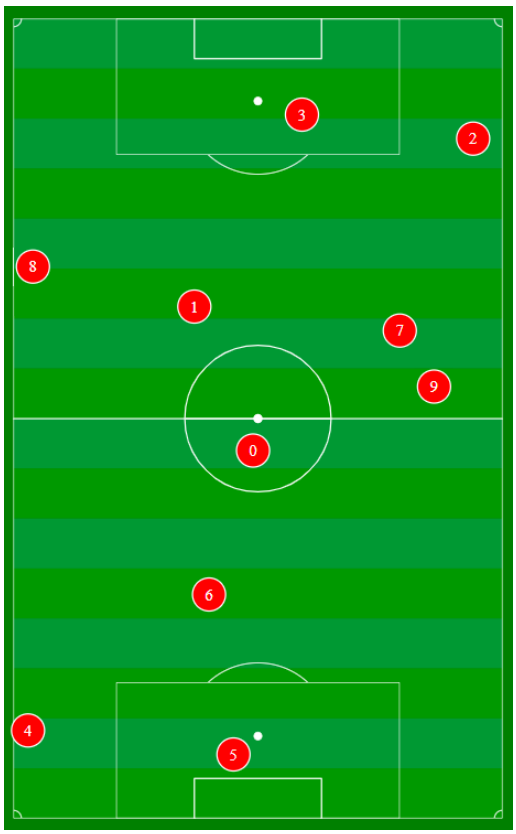
Hierna is het alleen nog maar een kwestie om de bediening van deze timer naar de consumer toegankelijk te maken. Er zijn drie functionaliteiten die op deze manier aangeboden worden:

- Pauze: pauzeert de timer en onthoudt bij welk element er gepauzeerd is.
- Stop: stopt de timer en zorgt er voor dat alle getekende element op het veld verwijderd worden.
- Play: hervat of start de timer en zorgt dat er bij het juiste element verder wordt afgespeeld.

6.3.4.2 Action Replay Resultaat

In figuur 12 is het uiteindelijke resultaat te zien. De visualisatie ontvangt als input een lijst aan events, die stuk voor stuk worden afgespeeld. De replay kan worden gestart, gepauzeerd en worden gestopt. In dit geval zijn het posities van passes die na achter elkaar worden afgespeeld.

Figuur 12: Action Replay resultaat



Naast het afspelen van de events, konden de eerder gebouwde factories hergebruikt worden om de Action Replay elementen te creëren. Hierdoor kunnen de Action Replay-elementen makkelijk geconfigureerd worden op kleur, vorm, positie etc. Dit zorgt voor een hele snelle ontwikkeling van deze visualisatie, omdat er eerder geschreven functionaliteit kan worden hergebruikt in deze visualisatie.

6.3.5 Evaluatie iteratie 3

Aan het begin van de iteratie had ik ingeschat de volgende onderdelen af te krijgen:

- Dynamische select
- Grid plot (Requirement ID: 10)
- Action Replay (Requirement ID: 19)

Deze iteratie is eigenlijk perfect verlopen, alle onderdelen zijn af gekomen, zoals vooraf ingeschat. Het grootste gedeelte van deze iteratie is er aan de dynamische select gewerkt. Dit kwam doordat dit in combinatie met het publish-subscribe pattern gebouwd moest worden. Dit heeft een losgekoppeld, herbruikbaar element opgeleverd, die op alle visualisaties te gebruiken is. Deze dynamische select is inmiddels geïmplementeerd in die Field Locations en Field Lines visualisatie.

Ook is duidelijk geworden dat er in iteratie 1 de juiste abstractiekeuze gemaakt, door het hele tekengedeelte van de library in een aparte laag te isoleren. Dit is duidelijk geworden tijdens de bouw van de Action Replay en de Grid Plot visualisatie. Het kostte namelijk nagenoeg geen extra tijd om de visualisatie daadwerkelijk op te bouwen en te tekenen. Alle code uit eerdere iteraties, zoals de factories, konden hiervoor hergebruikt worden. Ik denk dat dit daarom een goed teken is voor de bouw van toekomstige visualisaties.

6.3.6 Demo iteratie 3

Tijdens de demo van de laatste iteratie werd duidelijk dat er af is wat af moest komen, dus er is vooral tevredenheid. Op de wens van een voetbal die getekend kan worden na, waren er verder niet echt veel andere wensen of opmerkingen van deze iteratie. De library lijkt daarom van voldoende kwaliteit.

7 Testen

Tijdens en na de implementatie heb ik mij bezig gehouden met het testen van de library. Uit de gesprekken met de stakeholders, bleek dat er behoefte was aan bewijsvoering van de werking van de library. Ik heb er dan voor gekozen om een implementatietest te houden nadat de library afgebouwd was. Daarnaast heb ik er voor gekozen om iedere iteratie unit tests te schrijven, om de kwaliteit van de nieuw geschreven code te testen. Alle beschrijvingen van testen, testcases en resultaten zijn vastgelegd in het testdocument (zie Bijlage 5).

7.1 Implementatietest

Allereerst wordt gekeken naar hetgeen wat er bereikt moet worden met de testen, nadat de library ontwikkeld was. Er is namelijk behoefte vanuit de stakeholders aan bewijsvoering van de werking van library. Het gaat hier om de functionaliteit van de library die voor de gebruiker van de library toegankelijk is. Hoe bewijs je of deze functionaliteit werkt? Door een aantal situaties te bedenken waarin de situatie een bepaald gedrag moet vertonen. Dit kan prima door het te testen.

Om een functionaliteit te testen, worden de volgende stappen genomen:

1. bedenk een bepaalde actie die de library kan uitvoeren;
2. beschrijf de stappen die nodig zijn om deze actie te voltooien;
3. beschrijf het verwachte resultaat;
4. beschrijf het uiteindelijke resultaat;
5. vergelijk of de deze resultaten overeenkomen;
6. als deze resultaten overeenkomen, dan is de test geslaagd.

Deze stappen zijn de basis voor hoe de implementatietest gaat verlopen en zijn verwerkt in de opgestelde testcases.

Is dit voldoende bewijsvoering van de werking van de library? Het gaat er namelijk om dat de library in verschillende omgevingen hetzelfde gedrag vertoond. Alleen dan pas kan er gesteld worden dat de library daadwerkelijk werkt zoals verwacht wordt. Daarom worden dezelfde testcases twee keer uitgevoerd, in de meest gebruikte softwareomgevingen binnen ORTEC Sports:

- AngularJS
- Angular 2+

9 van de 10 de producten van ORTEC Sports draaien in deze omgevingen, dus in dit geval zijn deze twee situaties voldoende toereikend om de werking van de library aan te tonen.

Tabel 14: Voorbeeld testcase

ID	8
Beschrijving	Test de pauze functionaliteit van de Action Replay visualisatie
Requirement ID	20
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd
Teststappen	1. Creëer een Action Replay visualisatie 2. Creëer een lijst aan events, die in de Action Replay getoond kunnen worden. 3. Set deze lijst als data voor de Action Replay Visualisatie 4. Start de replay 5. Pauzeer de replay bij event nr 5 6. Herstart de replay
Verwacht resultaat	Veld wordt getekend. Op het moment dat de replay wordt gestart, verschijnen de events stuk voor stuk op het veld. Als de Action Replay gepauzeerd wordt, dan blijft de replay gepauzeerd op het juiste event staan. Op het moment dat er weer start wordt gedrukt, gaat de Action Replay weer bij het juiste event verder.
Post-conditie	Timer is gepauzeerd bij het juiste element. Timer wordt hervat en gaat weer verder bij het juiste element.
Resultaat Angular 2+	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm. Wanneer er pauze wordt gedrukt, blijft de Action Replay bevroren totdat er weer start wordt gedrukt. Vervolgens hervat de Action Replay weer bij het juiste event.
Resultaat AngularJS	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm. Wanneer er pauze wordt gedrukt, blijft de Action Replay bevroren totdat er weer start wordt gedrukt. Vervolgens hervat de Action Replay weer bij het juiste event.
Fail/Pass	Pass
Aantekeningen	

In tabel 14 is een resultaat te zien van een uitgevoerde testcase. Hier wordt pauze functionaliteit van de Action Replay visualisatie getest. Op deze manier heb ik alle functionaliteiten die toegankelijk zijn voor de consumer los getest. Alle testcases zijn terug te vinden in Bijlage 5.

7.1.1 Resultaten implementatietest

Alle testen hebben hetzelfde verwachte resultaat opgeleverd in beide omgevingen. Dit is dan ook een goed teken dat de library functioneert zoals verwacht. Wel zijn er een paar opmerkingen genoteerd bij de testcases. Dit zijn opmerkingen over hoe bepaalde functies of parameters gebruikt worden en hoe dit eventueel aangepast kan worden in de toekomst. Zo is er bijvoorbeeld naar voren gekomen dat er behoefte is aan het automatisch aanpassen van de velddimensies, wanneer er een half veld gecreëerd worden i.p.v. een heel veld.

7.2 Unit tests

Tijdens de bouw van de library, zijn er tijdens iedere iteratie unit tests geschreven. Allereerst is de functionaliteit gebouwd, waarna de desbetreffende functionaliteit getest is.

Ik heb er voor gekozen na de bouw van een functionaliteit unit tests te schrijven en bijvoorbeeld niet test driven development. De reden hiervoor is dat de visuele feedback uit een visualisatie belangrijker is dan of de unit test wel of niet slaagt. De visuele feedback is uiteindelijk hetgeen wat bepaalt of een functionaliteit werkt of niet. De unit tests moeten niet leidend zijn voor de gebouwde functionaliteit.

In het geval van test driven, zijn de unit tests leidend i.p.v. de gebouwde functionaliteit. Dit is dus precies hetgeen wat ik niet wil bereiken. Om deze reden zijn tijdens de iteraties de volgende stappen aangehouden:

- bouw de functionaliteit;
- beoordeel of de visuele output werkt naar behoren;
- test individueel losse onderdelen van de totstandkoming van de visuele output.

7.2.1 Onderdelen die getest zijn

Nadat een functionaliteit gebouwd is, wordt als eerst de visuele output beoordeeld. Vervolgens worden alle componenten die een rol hebben in deze visuele output opgesplitst. De publiek toegankelijke functionaliteiten van deze componenten zijn de uiteindelijke onderdelen die getest worden.

Zo wordt er bijvoorbeeld getest of de het juiste type objecten gecreëerd worden door de factories, of dat de publish-subscribe componenten naar behoren werken. Deze functionaliteiten zijn dan weer onderdeel van de totale visuele output.

7.2.2 Voor- en nadelen unit tests

Een ondervonden nadeel van deze manier van testen is de onderhoudbaarheid van de geschreven testen. Zo heb ik ondervonden dat bij een bepaalde verandering in de code, een groot deel van de testen soms opnieuw aangepast moesten worden. Je zou kunnen stellen dat dit een nadeel is van unit testing op zichzelf, omdat de unit tests sterk gekoppeld zijn aan hetgeen wat zij testen.

Een groot voordeel wat ik heb ondervonden van unit tests is de betrouwbaarheid van individuele functionaliteit. Als een toekomstige ontwikkelaar een bepaalde verandering maakt aan de code, dan zullen er unit tests wel of niet falen. In het geval dat deze niet falen, kan er van uit worden gegaan dat de geteste functionaliteiten nog steeds werken. Als deze wel falen, dan kunnen er bugs worden herkend en opgelost. Dit levert een hogere betrouwbaarheid in de geschreven code.

7.2.3 Resultaten unit tests

Hieronder, in het kort, een aantal resultaten van de unit tests. De volledige resultaten in diepte, zijn terug te vinden in het testdocument (zie Bijlage 5).

Definities coverage:

- Function coverage: aantal functies dat is uitgevoerd door de unit tests.
- Statement coverage: aantal statements dat is uitgevoerd door de unit tests.
- Branch coverage: aantal branches (mogelijke paden binnen een statement) dat is uitgevoerd door de unit tests.
- Line coverage: aantal regels code dat is uitgevoerd door de unit tests.

Tabel 15: Resultaten unit tests

Total lines discovered	1579
Total lines coverage	1354 (85%)
Total statement coverage	1414/1643 (86.06%)
Total branches coverage	86/124 (69.35%)
Total functions coverage	305/363 (84.02%)

8 Productevaluatie

- ***Requirements document***

Aan het begin van de afstudeerperiode heb ik een requirements document opgesteld. Het doel van het opstellen van dit document was dan ook om een duidelijk startpunt te creëren voor de ontwikkeling van de library. Dit doel is zeker bereikt, aangezien er een makkelijk begin kon worden gemaakt.

Duidelijk is wel geworden dat het requirements document, naar mate de ontwikkeling de library vorderde, wel langzaam zijn waarde verloor. Dit komt vooral omdat dit document vooral op hoog niveau de requirements specificeert. Requirements die meer de diepte in gaan, zijn pas later, tijdens de ontwikkeling van de library ondervonden.

- ***Ontwikkelde library***

Allereerst valt er niet veel te zeggen of de library uiteindelijk de functionaliteit bevat die het moest bevatten. Dit komt omdat er geen harde eis is afgesproken met de opdrachtgevers welke requirements er hoe dan ook af moesten. Wel kan ik zeggen dat ik tevreden ben dat er uiteindelijk 4 nieuwe visualisaties te gebruiken zijn door deze library te integreren in een bestaand systeem.

Persoonlijk ben ik heel tevreden over de uiteindelijk staat van de code van de library. Naar mijn gevoel voldoet de library aan softwarekwaliteitseisen, zoals uitbreidbaarheid en onderhoudbaarheid. Dit bewijst zich door hoe makkelijk ik in iteratie 2 en 3 nieuwe visualisaties kon bouwen en toevoegen door de in iteratie 1 gecreëerde elementen te hergebruiken of uit te breiden. In de toekomst te ontwikkelen visualisaties zullen deze elementen ook kunnen hergebruiken of uitbreiden.

Ook kwam aan het einde van de afstudeerperiode naar voren dat er in de toekomst misschien behoefte is aan basketbal-analyse. Met de library is het heel simpel om bijvoorbeeld een basketbalveld te creëren. Zo zijn gebouwde vormen herbruikbaar in een basketbalveld, en leent de architectuur zich makkelijk om nieuwe vormen toe te voegen. Als ik dit soort business cases voorgeschoteld krijg, dan zie ik al snel dat de library zich hier goed voor leent. Dit is dus een perfect voorbeeld van dat de library daadwerkelijk uitbreidbaar is en gebruikt kan worden in de toekomst.

Van tevoren is gesteld dat het grootste probleem is dat het veel ontwikkelingstijd kost om nieuwe visualisaties te creëren in bestaande producten. Dit probleem is dan ook opgelost: met een paar simpele regels code, kan er nu makkelijk een visualisatie gecreëerd worden. Daarnaast zijn de visualisaties makkelijk configureerbaar, zoals kleuren of vormen van elementen op het veld.

De library is op dit moment al in gebruik in een bestaand product waar er positiedata van spelers op een veld getoond wordt. Dat mijn library nu al gebruikt wordt in bestaande producten, is een bevestiging van dat de library zijn nut bewijst. De desbetreffende visualisatie was dan ook binnen een hele korte tijd gemaakt en geïntegreerd, waar dit normaal een veel langere tijd kost om dit te ontwikkelen.

Er is een library gecreëerd die het makkelijk maakt om visualisaties te creëren in bestaande producten. Daarnaast is het ook in de toekomst makkelijk om nieuwe visualisaties, zoals bijvoorbeeld basketbalvelden, toe te voegen. Je kunt dus wel stellen dat de doelstelling van de afstudeeropdracht (hoofdstuk 3.3) hiermee is bereikt.

- ***Testdocument***

Aan het begin van de afstudeerperiode werd gesteld dat er nadat de library gebouwd was, er behoefte was aan bewijsvoering van de werking van de library. Uiteindelijk heb ik er voor gekozen om dit d.m.v. testen te bereiken.

Het doel om bewijsvoering van de werking van de library is met behulp van dit document bereikt. Een belangrijke kanttekening is wel, dat er voldoende op- of aanmerkingen zijn op de library, die tijdens de implementatietest duidelijk werden. Een voordeel is dat dit nu wel gedocumenteerd is en een volgende ontwikkelaar hiermee aan de slag zou kunnen.

Een andere kanttekening is dat vooral de zogeheten ‘happy flow’ is getest. Zo houdt bijvoorbeeld iedere visualisatie een aantal stappen aan om een visualisatie te creëren, data toe te voegen en vervolgens te tekenen. Ondanks dat dit gedocumenteerd is bij de import van de library, wat gebeurt er als deze volgorde niet wordt gehandhaafd? Hoe gaan de visualisaties hiermee om? Dit zijn vraagstukken die niet getest zijn, omdat dit niet rechtstreeks bijdraagt tot het doeleinde wat met de testen bereikt moest worden.

Verder bevat het testdocument resultaten van de unit tests. Dit dient alleen als documentatie voor toekomstige ontwikkelaars, om te kunnen zien wat en hoe er momenteel getest is. Naast dat er een hoge test coverage behaald is, valt er verder niks over te zeggen. De resultaten zeggen niks over of de code daadwerkelijk werkt, het zegt alleen dat de geteste functionaliteit werkt. Dit dient alleen als bewijs voor betrouwbaarheid van de geschreven code, waarmee gedeeltelijk code-kwaliteit wordt aangetoond.

9 Procesevaluatie

- ***Planning***

Aan het begin van de afstudeerperiode heb ik er voor gekozen om een bepaalde planning te hanteren. Een planning blijft natuurlijk een inschatting, maar deze inschatting klopt redelijk. Alle geplande fases zijn op tijd doorlopen en er is nagenoeg geen uitloop geweest. Er is wel een kleine uitloop van een aantal dagen geweest op de ontwikkeling van de library, maar dit is logisch over een proces van 85 dagen.

De kleine uitloop op mijn planning die ik heb ondervonden is vooral ontstaan door zaken zoals het werken aan het verslag of andere onverwachte gebeurtenissen op kantoor, zoals bepaalde bedrijfsevenementen. Dit zijn zaken waar ik van tevoren niet op heb gerekend.

Volgende keer zou ik misschien iets meer rekening moeten houden met dit soort zaken. Zo zou ik bijvoorbeeld een buffer kunnen inplannen voor onverwachte gebeurtenissen en niet alle dagen volplannen met werkzaamheden.

- ***Gefaseerd werken***

Ik heb er voor gekozen om het afstudeertraject op te delen in fases en deze stap voor stap te doorlopen. Dit was een hele nuttige manier van werken, omdat er volledig naar bepaald doel of resultaat gewerkt kon worden. Op deze manier is eigenlijk de gehele afstudeeropdracht opgedeeld in meer behapbare delen. Dit heeft voor mij vooral voor meer overzicht gezorgd en het gevoel dat ik mij op een doel kon focussen i.p.v. een lang proces waar geen einde in zicht is.

Het nadeel van gefaseerd werken is dat er niet meer gewerkt kan worden aan producten die in een vorige fase zijn opgeleverd. Een voorbeeld is het requirements document. Dit document is tijdens de ontwikkeling van de library steeds minder van belang geworden, omdat er nieuwe requirements tijdens de ontwikkeling duidelijk werden.

- ***Iteratief ontwikkelen***

Ik heb er voor gekozen om iteratief de library te ontwikkelen. Dit heeft zich uitbetaald in dat er veel requirements naar voren zijn gekomen tijdens het ontwikkelproces. Hierdoor is een product ontstaan wat nauwer aansluit bij de verwachtingen van de stakeholders.

Ik heb er voor gekozen om in iteraties van 3 weken te werken, met als gedachte om veel functionaliteiten te kunnen afronden elke iteratie. Dit zou ik wellicht in een volgend traject anders aanpakken, met kortere iteraties. Ik zou dit eerder naar 2 weken veranderen, omdat dit zorgt dat er iets meer behapbare functionaliteit afkomt tijdens een iteratie. Hierdoor wordt het probleem in nog kleinere stukjes opgedeeld en is er ook meer ruimte voor feedback op functionaliteit op een lager niveau.

10 Bewijsvoering beroepstaken

- ***Analyseren probleemdomein & opstellen probleemstelling***

In hoofdstuk 2 wordt het probleemdomein geanalyseerd, namelijk de huidige situatie van ORTEC Sports. Het gaat hier het bedrijfsprofiel, de producten en de betrokkenen.

In hoofdstuk 3.2 wordt een probleemstelling opgesteld.

- ***Kritisch onderzoeken en methodisch werken***

In hoofdstuk 4 wordt een aanpak opgesteld om methodisch tot een eindresultaat te komen. Hoofdstuk 4.1 beschrijft een planning en een projectmanagementmethode: gefaseerd werken. In hoofdstuk 4.4 wordt beschreven hoe er iteratief aan de ontwikkeling van een softwareproduct wordt gewerkt.

- ***Leren leren: voorbereiden op volgende studiefase en beroep***

In hoofdstuk 2.5 zijn de technieken beschreven die gehanteerd worden binnen ORTEC. D3.js is een veelgebruikte library binnen ORTEC om datavisualisaties mee te creëren, waar ik nog geen ervaring mee had. Door dit te leren tijdens mijn afstudeerstage, blijf ik bij in mijn vak- en potentieel werkgebied.

- ***Realiseren van software***

In hoofdstuk 6 worden belangrijke beslissingen in de totstandkoming van de library beschreven. Deze library is een gerealiseerd stuk software. Het uiteindelijk resultaat wordt geëvalueerd in hoofdstuk 8.

- ***Vergaren en analyseren van requirements***

In hoofdstuk 5 wordt de totstandkoming van de requirements beschreven. Het gaat hier om requirements vergaren, analyseren en documenteren. Deze requirements zijn gedocumenteerd in een apart requirements document (zie Bijlage 2).

- ***Testen***

In hoofdstuk 7 wordt de totstandkoming en resultaten van de implementatietest en unit tests beschreven.

In Bijlage 5, ook wel testdocument genoemd, zijn de resultaten van de unit tests en de implementatie test te vinden.

11 Verklarende begrippenlijst

- ***Library***

Verzameling code die door programma's heen kunnen worden gebruikt.

- ***Framework***

Een geheel van softwarecomponenten dat kan worden gebruikt bij het programmeren van applicaties.

- ***Requirement***

Een enkelvoudige gedocumenteerde bepaling, wat een bepaald product of dienst zou moeten doen.

- ***Unit test***

Test die een softwaremodule of broncode afzonderlijk test.

- ***Scalable Vector Graphic (SVG)***

Standaard voor de beschrijving van grafische elementen die samen een tekening vormen.

- ***Single Page Application (SPA)***

Webapplicatie of website die past op een enkele webpagina. In een SPA wordt alle benodigde code opgehaald met een enkele laadactie, of de noodzakelijke middelen worden dynamisch geladen wanneer ze nodig zijn.

- ***Angular Component***

Individuele bouwsteen van een Angular webapplicatie. Een webapplicatie gebouwd in het Angular-raamwerk is een set van components en andere Angular-elementen.

- ***Web Components***

Een set van verschillende technologieën waarmee een ontwikkelaar herbruikbare web componenten kan creëren, waar de functionaliteit van deze componenten geïsoleerd is.

- ***Consumer van een library***

Element in een softwareprogramma die de library importeert en gebruikt.

- ***Design pattern***

Een generiek opgezette softwarestructuur, die een bepaald veelvoorkomend type software-ontwerpprobleem oplost.

- ***Event listeners***

Element in een softwareprogramma die wacht tot een bepaald type software evenement voorkomt.

- ***Publisher***

Element in een softwareprogramma die een bepaald type software evenement genereert of publiceert.

- ***Subscriber***

Element in een softwareprogramma die naar een bepaald type software evenement luistert of abboneert.

12 Literatuurlijst

- *Angular*. (z.d.). Geraadpleegd 20 februari 2019, van https://angular.io/
- *Angular - Angular Elements Overview*. (z.d.). Geraadpleegd 22 februari 2019, van https://angular.io/guide/elements
- *Angular - Overview of Angular Libraries*. (z.d.). Geraadpleegd 20 februari 2019, van https://angular.io/guide/libraries
- *Creating and publishing private packages | npm Documentation*. (z.d.). Geraadpleegd 10 mei 2019, van https://docs.npmjs.com/creating-and-publishing-private-packages
- *Custom Elements v1: Reusable Web Components | Web Fundamentals | Google Developers*. (z.d.). Geraadpleegd 20 maart 2019, van https://developers.google.com/web/fundamentals/webcomponents/customelements
- *D3 in Depth*. (z.d.). Geraadpleegd 26 maart 2019, van https://www.d3indepth.com/scales/
- *D3.js - Data-Driven Documents*. (z.d.). Geraadpleegd 5 februari 2019, van https://d3js.org
- *Design Patterns — A quick guide to Observer pattern.* (2019, 8 februari). Geraadpleegd 15 maart 2019, van https://medium.com/datadriveninvestor/design-patterns-a-quick-guide-to-observer-pattern-d0622145d6c2

- *How to Scale SVG | CSS-Tricks*. (2017, 27 juni). Geraadpleegd 18 maart 2019, van https://css-tricks.com/scale-svg/
- *Jasmine Documentation*. (z.d.). Geraadpleegd 22 maart 2019, van https://jasmine.github.io/
- *Observer vs Pub-Sub pattern*. (2018, 21 juli). Geraadpleegd 15 april 2019, van https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c?gi=4678bd15ec2e
- *Top Version Control Systems for Growing Teams | Perforce*. (z.d.). Geraadpleegd 22 maart 2019, van https://www.perforce.com/solutions/version-control
- *Verdaccio · A lightweight private npm proxy registry*. (z.d.). Geraadpleegd 10 mei 2019, van https://verdaccio.org/
- *viewBox*. (2019, 18 maart). [{websiteFull}]. Geraadpleegd 26 maart 2019, van https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/viewBox
- *Web Components*. (z.d.). Geraadpleegd 22 februari 2019, van https://developer.mozilla.org/en-US/docs/Web/Web_Components
- *What is Pub/Sub Messaging?* (z.d.). Geraadpleegd 15 maart 2019, van https://aws.amazon.com/pub-sub-messaging/

Bijlage

13 Bijlage inhoudsopgave

Bijlage 1: Requirements conflict

Inleiding

Door gesprekken te voeren is duidelijk geworden dat er twee verschillende verwachtingen zijn van de functionaliteit van de library. Duidelijk is geworden dat hier conflicten tussen bestaan. Dit document beschrijft de verwachtingen en de conflicten daartussen. Allerlaatst wordt er een advies gegeven over wat de oplossing zou kunnen zijn. Dit advies wordt voorgelegd aan de stakeholders, waardoor er een besluit genomen kan worden.

Verwachting stakeholder 1: Bertus Talsma

De eerste verwachting is een library waarin er voltooide visualisaties worden aangeboden, zoals een staafdiagram of taartdiagram. Er is dan de wens dat deze als componenten worden aangeboden waaraan configuratie en datasets kan worden geleverd als input, waarna de component dit op basis hiervan visualiseert.

Verwachting stakeholder 2: Freek van Buel

De tweede verwachting is een library waarin er verschillende visualisaties, zoals lijnen, cirkels en vierkanten makkelijk gebruikt kunnen worden. Voorbeeld: in HTML de mogelijkheid om te kunnen zeggen: ik wil op XY-coördinaat een lijn die loopt naar een ander XY-coördinaat. Op deze manier kunnen makkelijk componenten zoals bijvoorbeeld voetbalveldvisualisaties worden opgebouwd. Kortgezegd: een gesimplificeerde versie van d3.js.

Conflicten

Duidelijk is dat er twee verwachtingen zijn. Dit zijn compleet verschillende ideeën van wat de functionaliteit van de library inhoudt. De ene verwachting is een library waar er standaard visualisaties, zoals bijvoorbeeld staafdiagrammen, worden aangeboden. De andere verwachting is een library waar meer generieke visualisaties worden aangeboden, zoals lijnen, vierkanten en cirkels etc.
Er is een conflict in de behoefte die de functionaliteit van de library moet vervullen:

- Standaard componenten(staafdiagrammen/heatmaps) vs generieke componenten(lijnen/vierkanten/dots)?

Advies

Een library waarin standaard visualisaties worden aangeboden is eigenlijk een soort tweede versie van bijvoorbeeld chart.js, alleen dan met de visualisaties die ORTEC Sports graag wil gebruiken. Dit wordt een specifieke library die een encapsulatie van verschillende grafieken aanbiedt.
Een library die generieke visualisatie zoals lijnen e.d. aanbiedt, is veel meer herbruikbaar in iedere situatie. Elke lijn, cirkel of vierkant is dus een encapsulatie van de daar voor nodige d3-code. Hiermee kun je ook staafdiagrammen, voetbalvelden etc. mee opbouwen zonder d3-code te hoeven schrijven. Je biedt hier mee d3-functionaliteit aan in componenten.

Mijn advies is dan ook om voor meer generieke visualisaties te gaan om de volgende redenen:

- Biedt een gesimplificeerde versie aan van d3-functionaliteiten.
- Mogelijkheid om elk soort visualisatie/grafiek op te bouwen door deze componenten te gebruiken. Voorbeeld: een voetbalveld is een vierkant waarin er dan componenten zoals cirkels getekend kunnen worden.
- Daadwerkelijk in iedere situatie waarin er iets getekend/gevisualiseerd moet worden, is deze library te gebruiken: dus daadwerkelijk herbruikbaar.
- Dit is niet alleen nuttig voor ORTEC Sports-visualisaties, maar ook voor andere afdelingen die data willen visualiseren.
- Niet specifieke componenten (staafdiagram, heat map etc) aanbieden, maar de bouwstenen hiervoor: meer controle over elk onderdeel van een visualisatie.

Een meer generieke library is een veel krachtigere tool, die veel meer mogelijkheden met zich meebrengt en dus ook de mogelijkheid om specifieke ORTEC Sports visualisaties te maken.

Bijlage 2: Requirements document

1 Inleiding

Onderdeel van de afstudeerperiode is het verzamelen en documenteren van requirements. Aangezien dit een tussenproduct is van de afstudeeropdracht, is er gekozen deze requirements vast te leggen in dit requirements document.

Dit document heeft als doel inzicht te geven in de globale/high level requirements, zodat er een duidelijk startpunt gecreëerd kan worden om aan het ontwerp en bouw van de library te beginnen. Ook is dit document nuttig als validatiemiddel m.b.t. de verwachtingen van de stakeholders. Het gaat hier nog niet om gedetailleerde requirements, die worden namelijk pas duidelijk tijdens het ontwerpen en bouwen van de library.

Het document is onderverdeeld in niet-functionele en functionele requirements. Verder zijn de functionele requirements onderverdeeld in type visualisatie.

Per requirement worden de volgende onderdelen beschreven:

- Een ID ter identificatie.
- Een korte beschrijving van wat de requirement inhoudt.
- Type requirement:
    - Business
    - Systeem
    - Gebruiker
- Functioneel of niet-functioneel.
- Prioriteit op basis van de MoSCoW-methode:
    - Must haves: moeten in het eindproduct terugkomen, anders is het product niet bruikbaar.
    - Should haves: zeer gewenst, maar zonder is het product nog steeds bruikbaar.
    - Could haves: komen alleen aan bod als er tijd genoeg is.
    - Would haves: komen waarschijnlijk niet aan bod, maar kunnen in de toekomst interessant zijn.
- Business owner.
- Status ter validatie: goedgekeurd of open.

2 Niet functionele requirements

ID	1
Titel	Library bevat componenten die data visualiseren.
Beschrijving	Library bevat componenten die datasets die als input nodig hebben, waarna deze worden omgezet naar een visualisatie.
Type	Systeem requirement
Functioneel/niet functioneel	Niet functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	2
Titel	Library werkt met verschillende datasets.
Beschrijving	Elke component in de library werkt met datasets als input. Datasets worden volgens een bepaald format aan elke component gegeven.
Type	Systeem requirement
Functioneel/niet functioneel	Niet functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Arjan Peters
Status	Goedgekeurd

ID	3
Titel	De library wordt gebouwd in Angular 7
Beschrijving	Angular is een techniek die binnen ORTEC als standaard gehanteerd wordt. De library wordt dan ook gebouwd in de laatste versie van Angular, versie 7.2.4.
Type	Systeem requirement
Functioneel/niet functioneel	Niet functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Arjan Peters
Status	Goedgekeurd

ID	4
Titel	Componenten in Angular 7 worden ook gecompileerd naar Web Components.

Beschrijving	Componenten in Angular 7 worden gecompileerd naar Angular elements, ook wel Web Components genoemd.
Type	Systeem requirement
Functioneel/niet functioneel	Niet functioneel
Acceptatiecriteria	Componenten moeten ook in Angular 1.6.x werken.
Prioriteit	Should have
Business owner	Arjan Peters
Status	Goedgekeurd

3 Functionele requirements

3.1 Veldvisualisatie

ID	5
Titel	De library bevat een veldvisualisatie
Beschrijving	De library bevat een visualisatie van een voetbalveld, waarop data getekend kan worden.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	6
Titel	Een veldvisualisatie kan geconfigureerd worden als een heel of een half veld
Beschrijving	Er kan geconfigureerd worden of er een heel of half held getoond wordt.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Should have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.2 Passmap

ID	7
Titel	De library bevat een passmap
Beschrijving	De library bevat een visualisatie van een passmap, waarop passes tussen spelers worden gevisualiseerd op basis van geleverde input.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	8
Titel	Bij een passmap kunnen de kleuren van de passlijntjes, spelersbolletjes en de lijndikte geconfigureerd worden.
Beschrijving	Bij een passmap kunnen kleuren ingesteld worden, bijvoorbeeld voor elk team een andere kleur. Daarnaast moet het mogelijk zijn om te kunnen instellen na hoeveel passes een lijntje dikker wordt.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	9
Titel	Bij een hover over een speler in een passmap, worden alleen de passes van en naar die desbetreffende speler getoond.
Beschrijving	Een gebruiker moet de mogelijkheid hebben om de passes van een individuele speler te kunnen bekijken. Dit wordt mogelijk gemaakt door met de muis over een speler heen te bewegen, waarna de passes van en naar deze speler gefilterd en getekend worden.
Type	Gebruikers requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.3 Grid plot

ID	10
Titel	Een veldvisualisatie bevat een een grid, die dynamisch is opgebouwd
Beschrijving	Op een veldvisualisatie worden vakjes getekend op basis van gegeven input. Input wordt geleverd vanuit de consumer van de library. Op deze vakjes kan data gevisualiseerd worden zoals een aggregatie van acties op het veld in zones.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	11
Titel	Een vakje bevat een tekstveld waarop data kan worden getoond.
Beschrijving	Op elk vakje kan data als input worden meegegeven, die daarop getoond wordt.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	12
Titel	Het aantal vakjes op een gridplot kan worden geconfigureerd.
Beschrijving	Elk gridplot moet anders kunnen worden weergegeven. Zo is er bijvoorbeeld een andere verdeling van vakjes op het veld bij cornervisualisaties dan bij passvisualisaties.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Should have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.4 Field average

ID	13
Titel	De library bevat een field average visualisatie.
Beschrijving	Op een field average worden het gemiddeld aantal passes per richting gevisualiseerd d.m.v. pijlen.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	14
Titel	Bij een field average visualisatie, is er de mogelijkheid om de kleur van de pijlen de configureren.
Beschrijving	Op een field average worden het gemiddeld aantal passes per richting gevisualiseerd d.m.v. pijlen.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Should have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.5 Field lines

ID	15
Titel	De library bevat een field lines visualisatie.
Beschrijving	Een field lines grafiek visualiseert acties en de bijbehorende richting in punten en lijnen.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	16
Titel	Een hover op een actiepunt toont een tooltip met bijbehorende data.
Beschrijving	De gebruiker moet de mogelijkheid hebben om als hij met zijn muis over een actiepunt heengaat, bijbehorende info van dat actiepunt te kunnen bekijken. Deze info wordt getoond in een tooltip.
Type	Gebruikers requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Should have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.6 Field locations

ID	17
Titel	De library tekent op een veldvisualisatie punten.
Beschrijving	Op basis van data, die als input gegeven wordt, wordt er op de veldvisualisatie punten getekend.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	18
Titel	De punten op het veld zijn configureerbaar.
Beschrijving	De punten worden geconfigureerd door de consumer van de library. Het gaat hier om configuratie zoals vorm, kleur, positie.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Acceptatiecriteria	
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.7 Action replay

ID	19
Titel	De library bevat een action replay visualisatie.
Beschrijving	Een action replay visualiseert een herhaling van een bepaalde actie in een wedstrijd.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Must have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	20
Titel	Een gebruiker kan de action replay pauzeren en herstarten
Beschrijving	De gebruiker heeft de mogelijkheid d.m.v. knoppen een action replay te pauzeren of opnieuw te bekijken.
Type	Gebruikers requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Should have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	21
Titel	De action replay visualisatie genereert meldingen wanneer er belangrijke events plaatsvinden in de visualisatie.

Beschrijving	Als er belangrijke events, zoals bijvoorbeeld goals plaatsvinden, dan genereert de library een melding.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Could have
Business owner	Bertus Talsma
Status	Goedgekeurd

ID	22
Titel	De action replay visualisatie genereert een overzicht van betrokken spelers, op basis van data die als input geleverd wordt.
Beschrijving	Er wordt als input een dataset met events geleverd. Vanuit hier wordt de action replay gevisualiseerd. Ten behoeve van overzicht, worden de betrokken spelers bij de desbetreffende action replay getoond.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Could have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.8 Overige visualisaties

3.8.1 Spidergrafiek

ID	23
Titel	De library bevat een spidergrafiek.
Beschrijving	Een spidergrafiek visualiseert kenmerken van bijvoorbeeld een speler.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Could have
Business owner	Bertus Talsma
Status	Goedgekeurd

3.8.2 Scatterplot

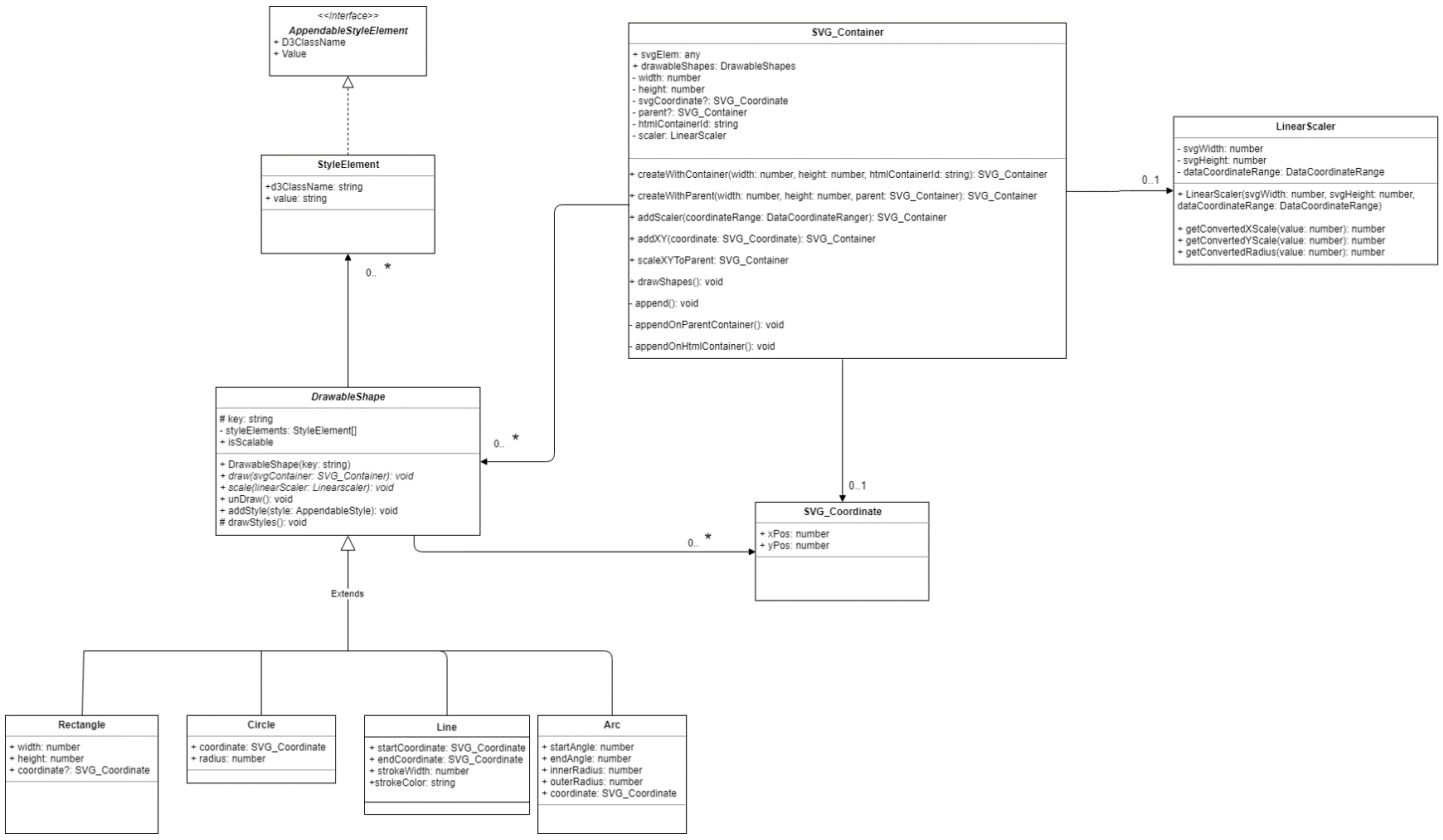
ID	24
Titel	De library bevat een scatterplot.
Beschrijving	Een scatterplot visualiseert de uitkomsten op twee variabelen.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Could have
Business owner	Bertus Talsma
Status	Goedgekeurd

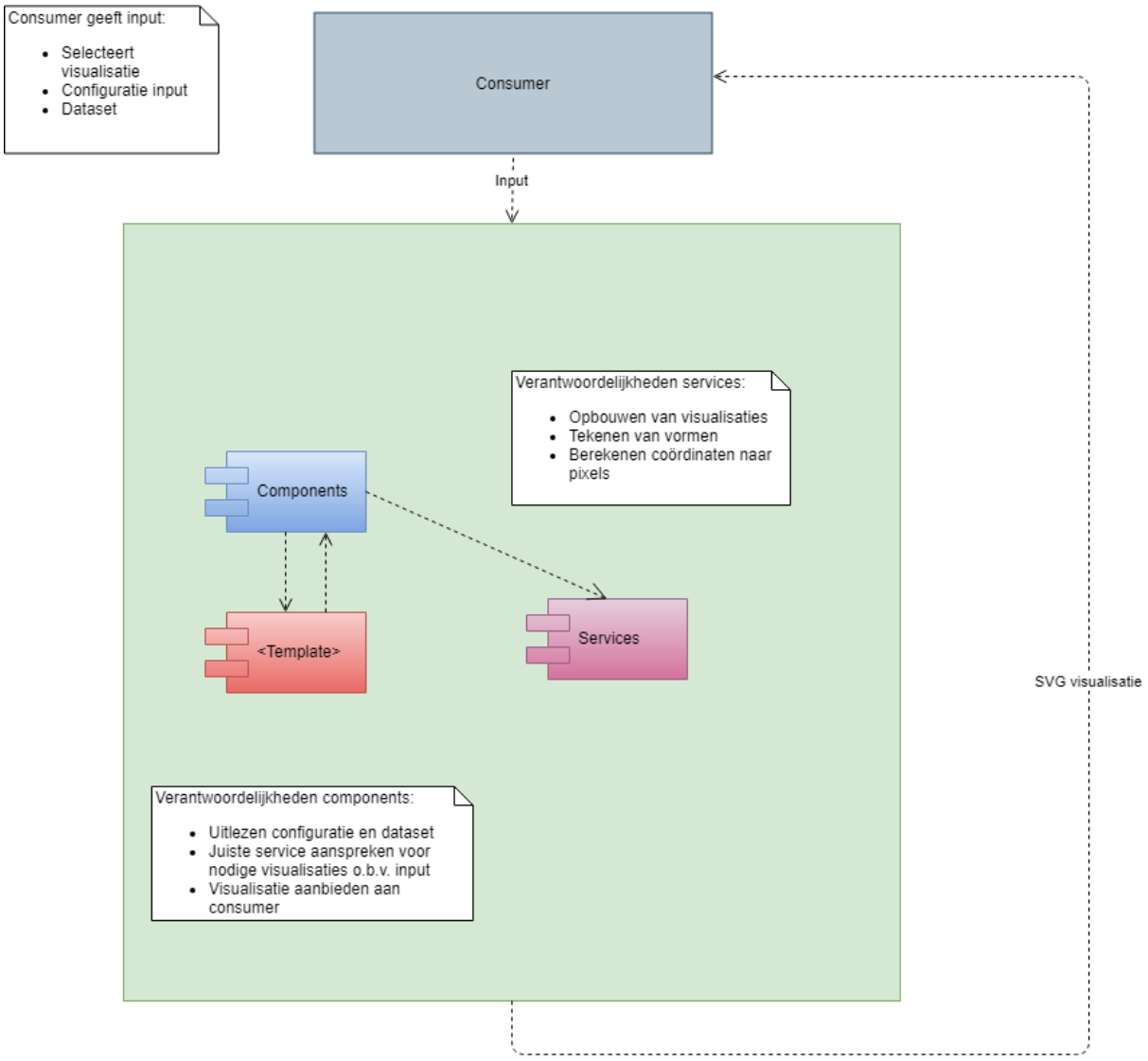
3.8.3 Lijngrafiek

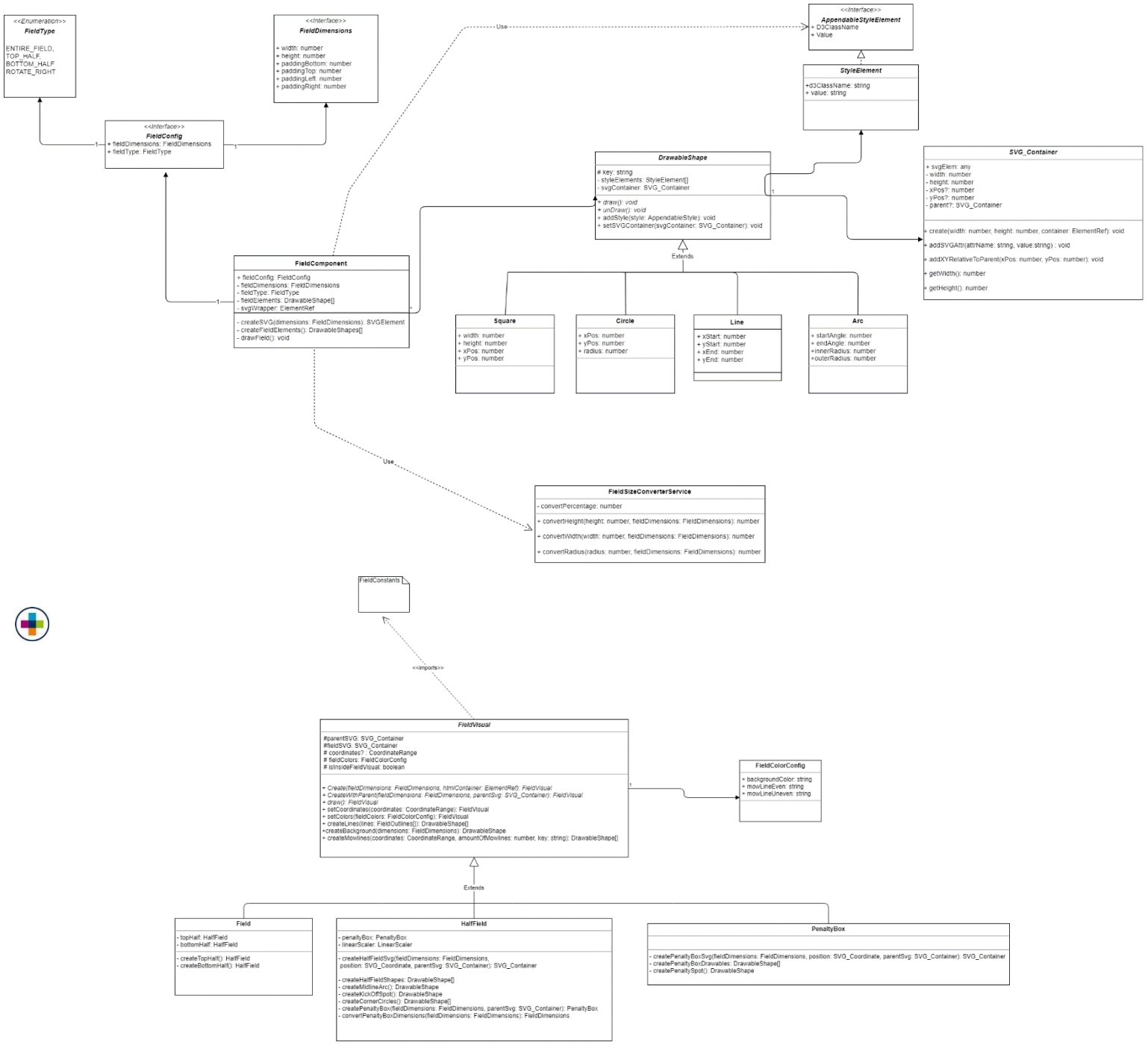
ID	25
Titel	De library bevat een lijngrafiek.
Beschrijving	Een lijngrafiek visualiseert progressie over tijd/rondes.
Type	Systeem requirement
Functioneel/niet functioneel	Functioneel
Prioriteit	Could have
Business owner	Bertus Talsma
Status	Goedgekeurd

Bijlage 3: Iteraties UML diagrammen

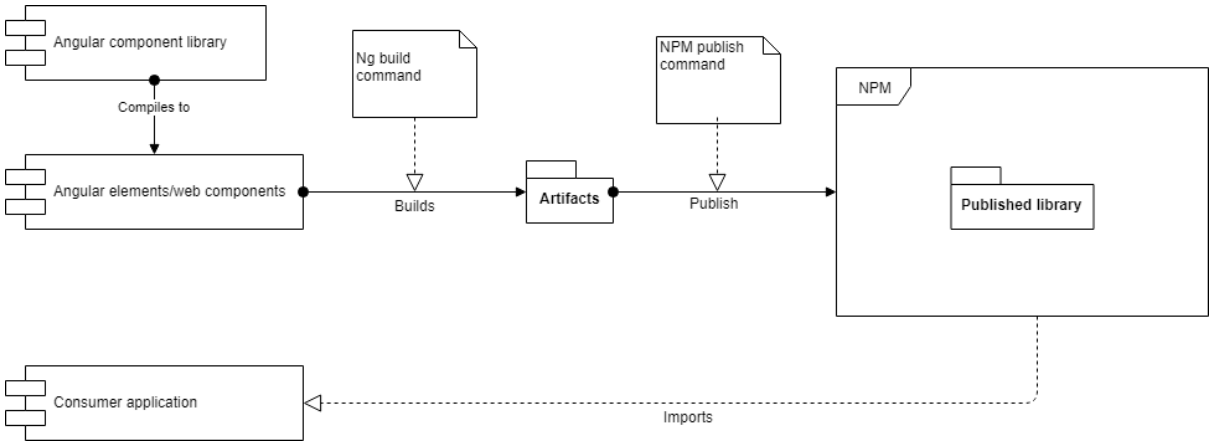
1. Iteratie 1

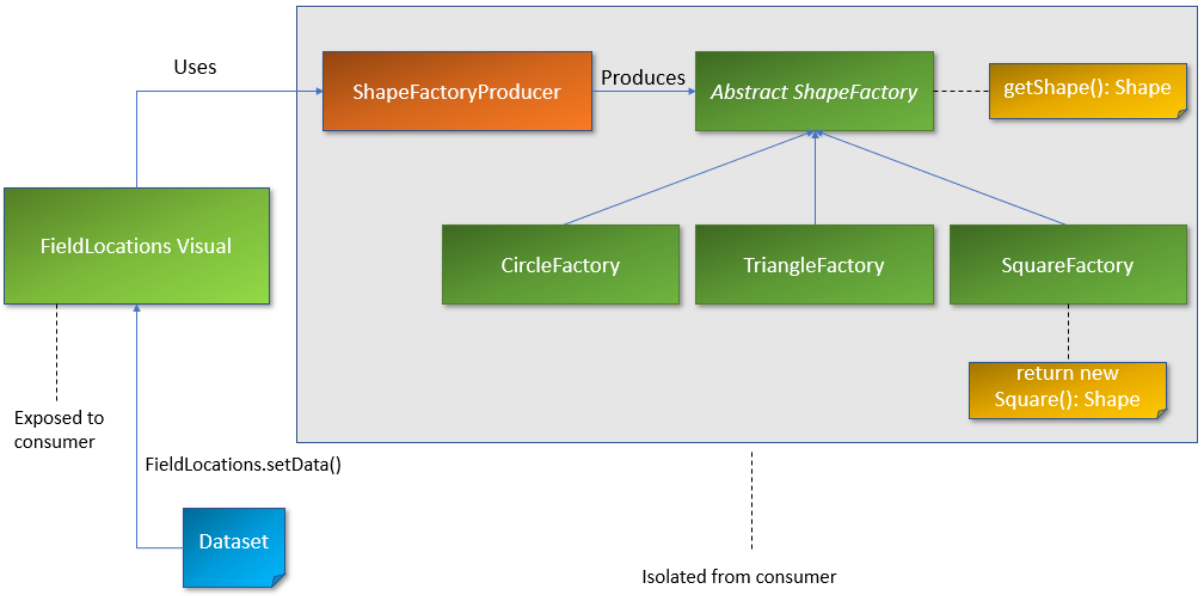



Consumer geeft input:
Selecteert visualisatie
Configuratie input
Dataset
Consumer
Input
Verantwoordelijkheden services:
Opbouwen van visualisaties
Tekenen van vormen
Berekenen coördinaten naar pixels
Components
<Template>
Services
SVG visualisatie
Verantwoordelijkheden components:
Uitlezen configuratie en dataset
Juiste service aanspreken voor nodige visualisaties o.b.v. input
Visualisatie aanbieden aan consumer

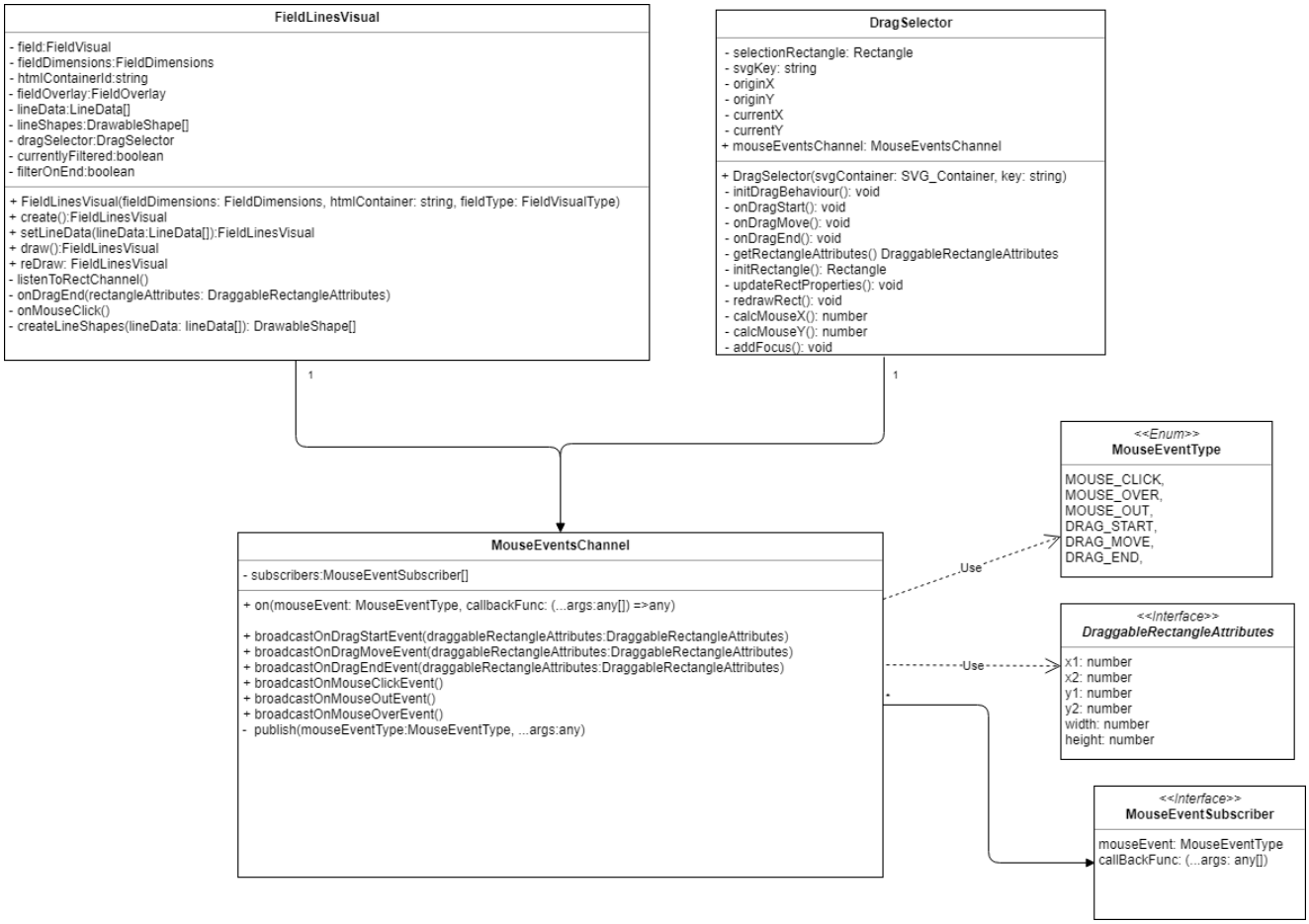

<<Enumeration>>
FieldType
ENTIRE_FIELD,
TOP_HALF,
BOTTOM_HALF
ROTATE_RIGHT
<<Interface>>
FieldDimensions
+ width: number
+ height: number
+ paddingBottom: number
+ paddingTop: number
+ paddingLeft: number
+ paddingRight: number
<<Interface>>
AppendableStyleElement
+ D3ClassName
+ Value
StyleElement
+d3ClassName: string
+ value: string
<<Interface>>
FieldConfig
+ fieldDimensions: FieldDimensions
+ fieldType: FieldType
Use
DrawableShape
key: string
- styleElements: StyleElement[]
- svgContainer: SVG_Container
+ draw(): void
+ unDraw(): void
+ addStyle(style: AppendableStyle): void
+ setSVGContainer(svgContainer: SVG_Container): void
SVG_Container
+ svgElem: any
- width: number
- height: number
- xPos?: number
- yPos?: number
- parent?: SVG_Container
+ create(width: number, height: number, container: ElementRef): void
+ addSVGAttr(attrName: string, value:string) : void
+ addXYRelativeToParent(xPos: number, yPos: number): void
+ getWidth(): number
+ getHeight(): number
Extends
FieldComponent
+ fieldConfig: FieldConfig
- fieldDimensions: FieldDimensions
- fieldType: FieldType
- fieldElements: DrawableShape[]
- svgWrapper: ElementRef
- createSVG(dimensions: FieldDimensions): SVGElement
- createFieldElements(): DrawableShapes[]
- drawField(): void
Square
+ width: number
+ height: number
+ xPos: number
+ yPos: number
Circle
+ xPos: number
+ yPos: number
+ radius: number
Line
+ xStart: number
+ yStart: number
+ xEnd: number
+ yEnd: number
Arc
+ startAngle: number
+ endAngle: number
+innerRadius: number
+outerRadius: number
Use
FieldSizeConverterService
- convertPercentage: number
+ convertHeight(height: number, fieldDimensions: FieldDimensions): number
+ convertWidth(width: number, fieldDimensions: FieldDimensions): number
+ convertRadius(radius: number, fieldDimensions: FieldDimensions): number
FieldConstants
<<imports>>
FieldVisual
#parentSVG: SVG_Container
#fieldSVG: SVG_Container
coordinates? : CoordinateRange
fieldColors: FieldColorConfig
isInsideFieldVisual: boolean
+ Create(fieldDimensions: FieldDimensions, htmlContainer: ElementRef): FieldVisual
+ CreateWithParent(fieldDimensions: FieldDimensions, parentSvg: SVG_Container): FieldVisual
+ draw(): FieldVisual
+ setCoordinates(coordinates: CoordinateRange): FieldVisual
+ setColors(fieldColors: FieldColorConfig): FieldVisual
+ createLines(lines: FieldOutlines[]): DrawableShape[]
+createBackground(dimensions: FieldDimensions): DrawableShape
+ createMowlines(coordinates: CoordinateRange, amountOfMowlines: number, key: string): DrawableShape[]
FieldColorConfig
+ backgroundColor: string
+ mowLineEven: string
+ mowLineUneven: string
Extends
Field
- topHalf: HalfField
- bottomHalf: HalfField
- createTopHalf(): HalfField
- createBottomHalf(): HalfField
HalfField
- penaltyBox: PenaltyBox
- linearScaler: LinearScaler
- createHalfFieldSvg(fieldDimensions: FieldDimensions,
position: SVG_Coordinate, parentSvg: SVG_Container): SVG_Container
- createHalfFieldShapes: DrawableShape[]
- createMidlineArc(): DrawableShape
- createKickOffSpot(): DrawableShape
- createCornerCircles(): DrawableShape[]
- createPenaltyBox(fieldDimensions: FieldDimensions, parentSvg: SVG_Container): PenaltyBox
- convertPenaltyBoxDimensions(fieldDimensions: FieldDimensions): FieldDimensions
PenaltyBox
- createPenaltyBoxSvg(fieldDimensions: FieldDimensions, position: SVG_Coordinate, parentSvg: SVG_Container): SVG_Container
- createPenaltyBoxDrawables: DrawableShape[]
- createPenaltySpot(): DrawableShape

2. Iteratie 2


Angular component library
Compiles to
Angular elements/web components
Ng build command
Builds
Artifacts
NPM publish command
Publish
NPM
Published library
Consumer application
Imports


Uses
ShapeFactoryProducer
Produces
Abstract ShapeFactory
getShape(): Shape
FieldLocations Visual
CircleFactory
TriangleFactory
SquareFactory
return new Square(): Shape
Exposed to consumer
FieldLocations.setData()
Dataset
Isolated from consumer

3. Iteratie 3


FieldLinesVisual
- field:FieldVisual
- fieldDimensions:FieldDimensions
- htmlContainerId:string
- fieldOverlay:FieldOverlay
- lineData:LineData[]
- lineShapes:DrawableShape[]
- dragSelector:DragSelector
- currentlyFiltered:boolean
- filterOnEnd:boolean
+ FieldLinesVisual(fieldDimensions: FieldDimensions, htmlContainer: string, fieldType: FieldVisualType)
+ create():FieldLinesVisual
+ setLineData(lineData:LineData[]):FieldLinesVisual
+ draw():FieldLinesVisual
+ reDraw: FieldLinesVisual
- listenToRectChannel()
- onDragEnd(rectangleAttributes: DraggableRectangleAttributes)
- onMouseClick()
- createLineShapes(lineData: lineData[]): DrawableShape[]
DragSelector
- selectionRectangle: Rectangle
- svgKey: string
- originX
- originY
- currentX
- currentY
+ mouseEventsChannel: MouseEventsChannel
+ DragSelector(svgContainer: SVG_Container, key: string)
- initDragBehaviour(): void
- onDragStart(): void
- onDragMove(): void
- onDragEnd(): void
- getRectangleAttributes() DraggableRectangleAttributes
- initRectangle(): Rectangle
- updateRectProperties(): void
- redrawRect(): void
- calcMouseX(): number
- calcMouseY(): number
- addFocus(): void
1
1
MouseEventsChannel
- subscribers:MouseEventSubscriber[]
+ on(mouseEvent: MouseEventType, callbackFunc: (...args:any[]) =>any)
+ broadcastOnDragStartEvent(draggableRectangleAttributes:DraggableRectangleAttributes)
+ broadcastOnDragMoveEvent(draggableRectangleAttributes:DraggableRectangleAttributes)
+ broadcastOnDragEndEvent(draggableRectangleAttributes:DraggableRectangleAttributes)
+ broadcastOnMouseClickEvent()
+ broadcastOnMouseOutEvent()
+ broadcastOnMouseOverEvent()
- publish(mouseEventType:MouseEventType, ...args:any)
Use
Use
*
<<Enum>>
MouseEventType
MOUSE_CLICK,
MOUSE_OVER,
MOUSE_OUT,
DRAG_START,
DRAG_MOVE,
DRAG_END,
<<Interface>>
DraggableRectangleAttributes
x1: number
x2: number
y1: number
y2: number
width: number
height: number
<<Interface>>
MouseEventSubscriber
mouseEvent: MouseEventType
callBackFunc: (...args: any[])

Bijlage 4: initiële terugkoppeling requirements

Library

Data niveau Dimensie 1

1) Annotatie /Event -> Elke actie apart
2) Match -> Totalen per match op statistiek level
   Kan subfilters hebben zoals Helft of kwartier.
   Eerste niveau aggregatie Annotaties
3) Multiple matches (single observation per match)
   Reeks van waardes maar per wedstrijd
4) Multiple matches Aggregated (competitie)
   Meerder wedstrijden geaggregeerd tot 1 waarde (kan deel van competitie zijn

Data niveau Dimensie 2

- Speler
- Team
- Alle spelers Team
- Competitie
- Multi Competite

Type visualisaties

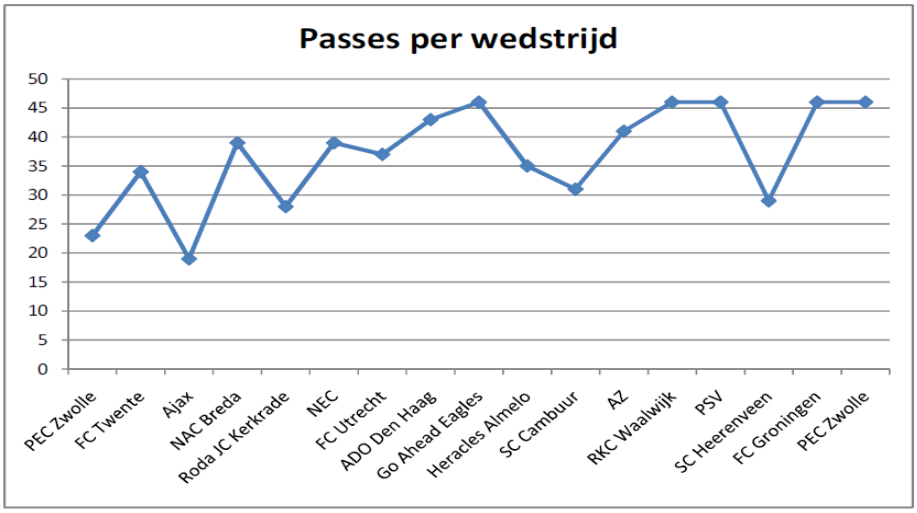
Type	Level	Input	Beschrijving doel
Field + point	Match Multi Match (low number)	5 Dim - X,Y -> Locatie -Type-> symbol - kleur -waarde -> size	Overzicht locatie van events
Field + point+endpoint	Match	Same as point + Xend, Yend -Waarde (size arrow)	Overzicht richting van actie
Field Grid	Match/MultiMatch/Comp	Number of gridblocks - start(X,Y) - end(X,Y) - Count - Count2	Overzicht van acties op veld geaggregeerd naar zones. Aggregratie binnen in de library of alleen de visualisatie?
Field Average	Multi Match Comp	Number of avergers -(X,Y) -Type -Value	Aggregatie van locatie waardes naar 1 gemiddelde locatie
Field Heatmap	Match->Comp	Raw (X,Y)	Transformatie van x,y naar density.
Line graph	Multi Match – Comp	Input - X values (categorical) - Y_1 values - Y_2 values	Progressie over tijd/roundes Tweede waarde als trend
Bar graph	Multi Match – Comp	Input - X values (categorical) - Y_1 values - Y_2 values	Vergelijking categorieën
Pie diagram	Comp	Input - Categorien - Waardes	Onderverdeling van subgroepen.

Scatterplot	**Comp**	**- X,Y -> Locatie -Type-> symbol (icon) - kleur -waarde -> size (zelfde als field plot)**	**Vergelijking van categorien op 2 of meerdere dimensies**
Spider graph	**Comp**	**Multi -Score -Type -Labels**	**Vergelijken van 2 of meerdere categorieen (spelers/teams) op basis van meerdere variabelen**
Box plot	**Multi match / Comp**	**Multi groups** - **Min** - **1th quartile** - **Average** - **3th quartile** - **Max** - **Outliers** - **Labels**	**Vergelijking van groupen + spreiding binnen groep.**

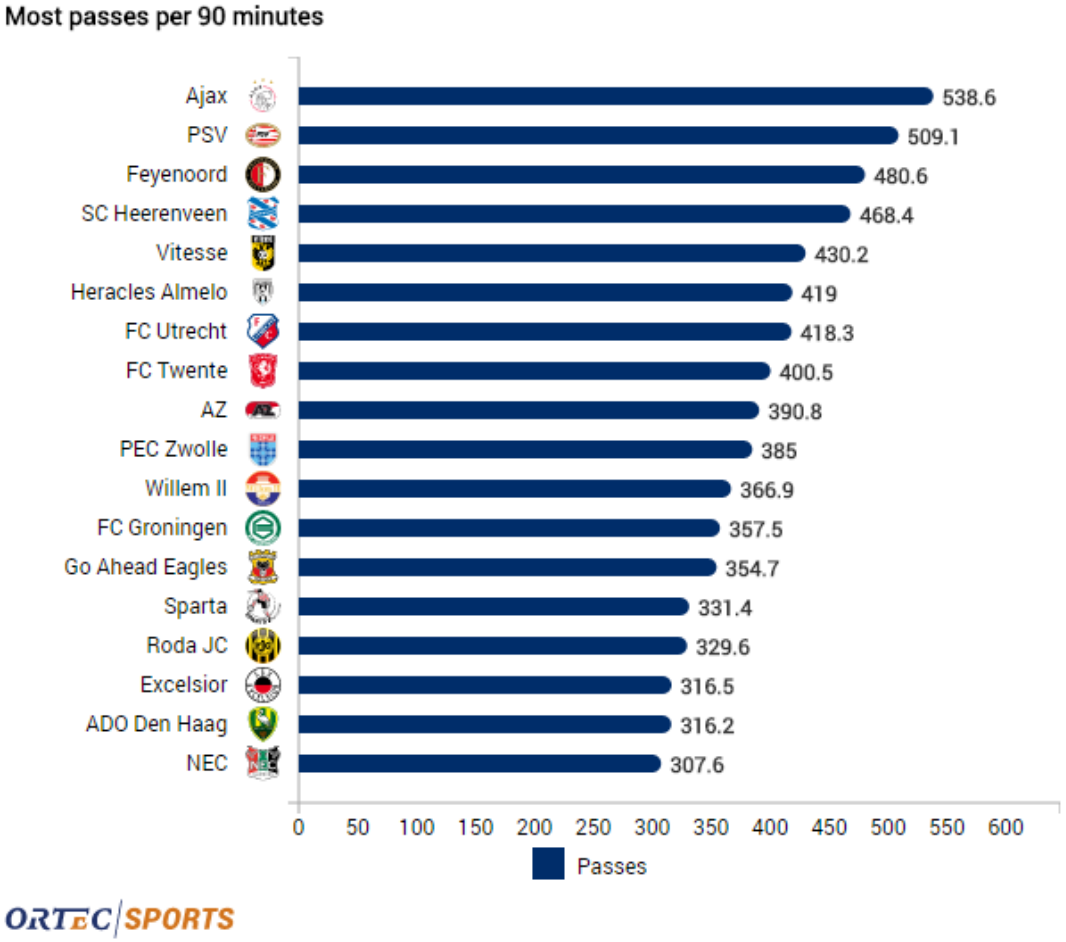
Data Niveau **Level**	**Event/Annotatie**	**Match**	**Competition Per match**	**Competion Aggregated**
Team	**Field plot point**	**Field Grid**	**Line graph** **Bar graph**	**Field Grid**
Player	**Field plot point**	**Field Grid**		**Field Grid** **Spidergraph**
Team all players	**Average Field plot**		**Average Field plot**	**Scatterplot**
Competition /Teams			**Line graph** **Bar graph**	**Scatterplot** **Box Plot**
Multi Comp				**Box Plot**

Examples

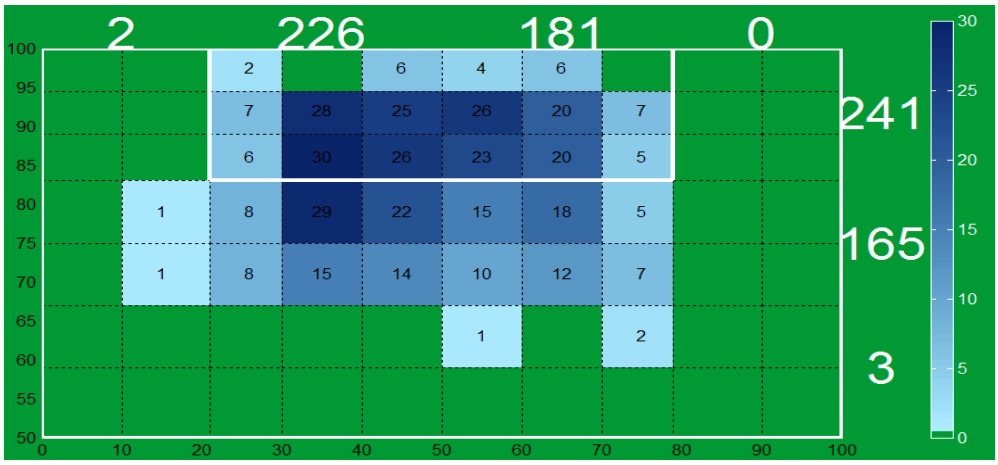
Line graph : Team score per Ronde, labels tegenstander



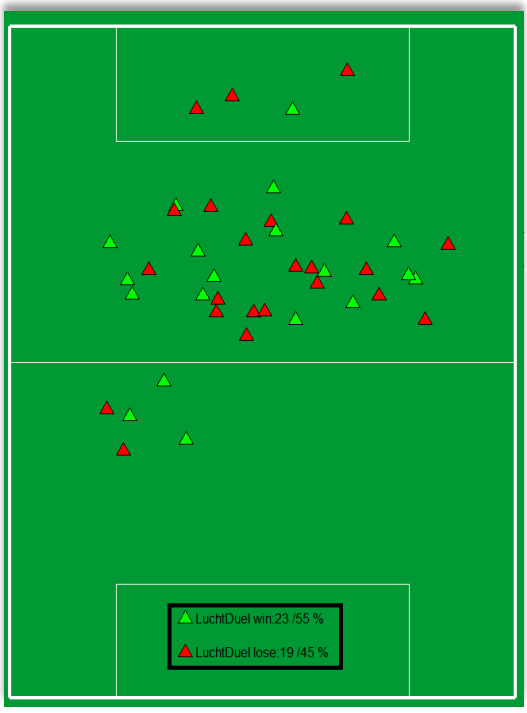
Bar graph : Team vergelijking 1 statistiek, Competitie.



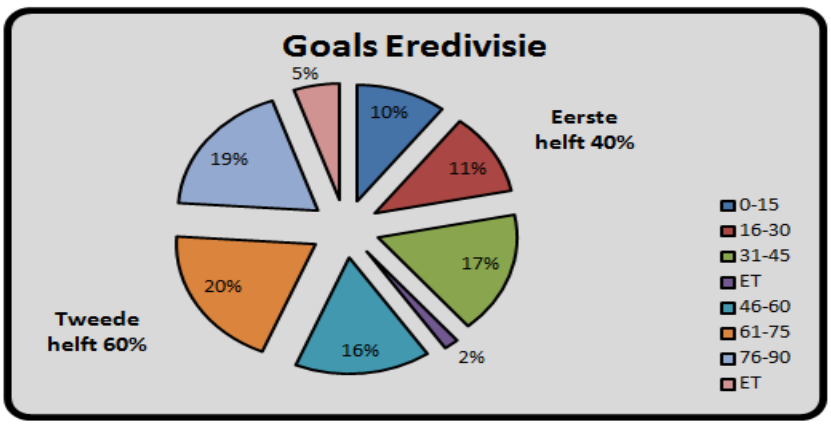
Field Grid: Team doelpogingen Competitie



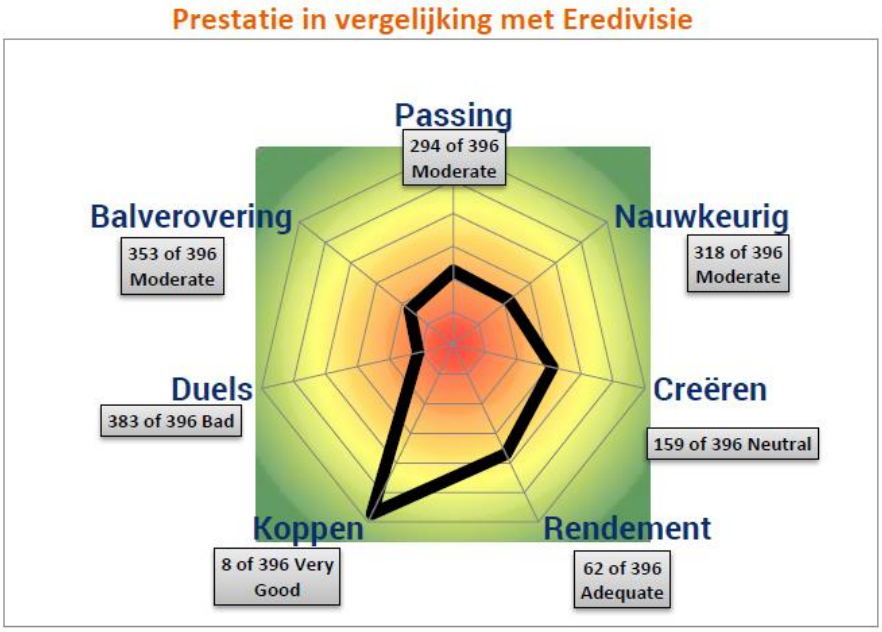
Field point : Team Match Duels



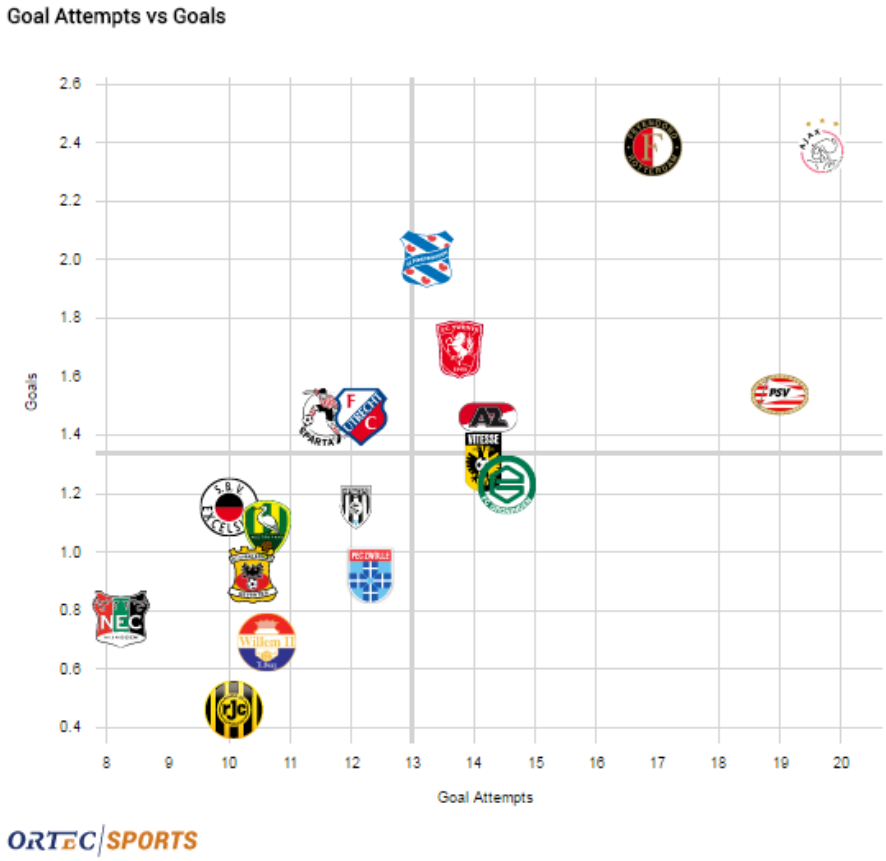
Pie diagram: Competitie : doelpunten ->Subgroeps kwarts.



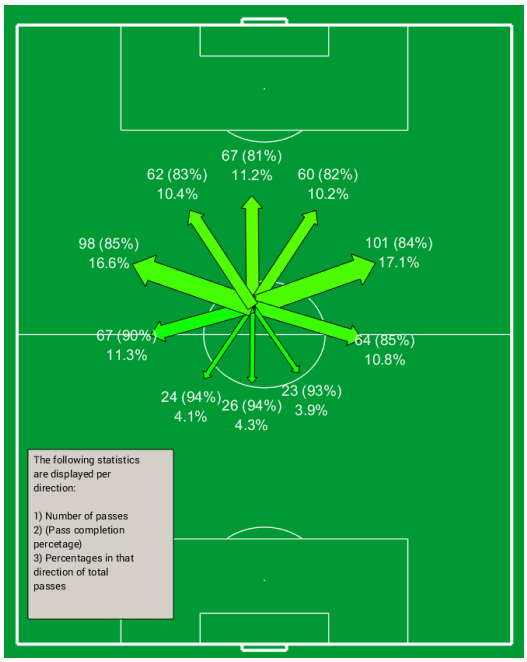
Spider plot: Speler prestatie



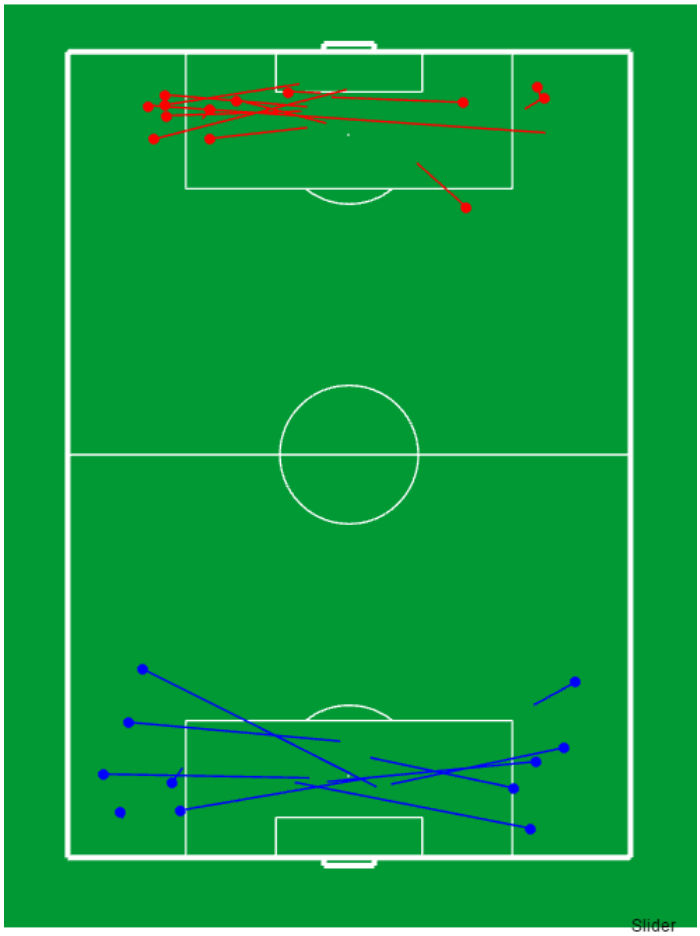
Scatterplot: 2 statistieken, Teams



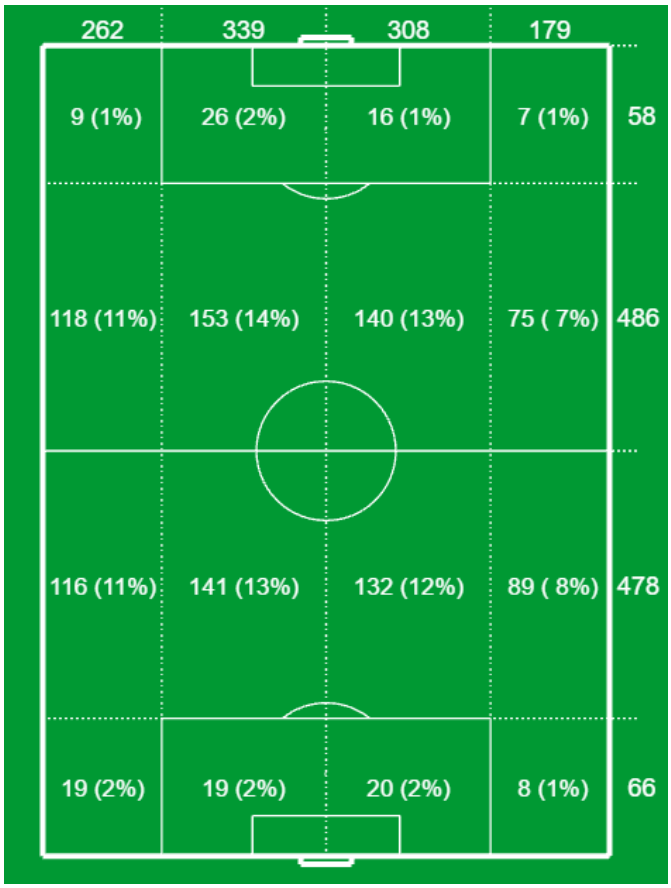
Field Average/Arrow Gemiddeld aantal passes per richting + pass completion (kleur).



Field line: Teams Voorzet richting.



Field Grid locatie



Bijlage 5: Testdocument

1 Inleiding

Onderdeel van de afstudeeropdracht Datavisualisatie, is het testen van de gebouwde datavisualisatie-library.

Dit document heeft als doel om bewijsvoering te leveren van de werking van de library. Dit doel wordt bereikt met het uitvoeren van testen de resultaten hiervan vast te leggen in dit document. Dit document bevat dan ook alleen de resultaten en niet de totstandkoming.

Dit document begint met de onderdelen beschreven die getest zijn, en de wijze waarop. Ook wordt de aanpak van de testen toegelicht. Hoofdstuk 2 bevat de uitgevoerde testcases en de resultaten hiervan. Allerlaatst bevat hoofdstuk 3 de resultaten van de uitgevoerde unit tests.

2 Onderdelen die getest worden

2.1 Functionaliteit van de visualisaties

Het belangrijkste onderdeel van de library wat getest moet worden is de werking van de visualisaties. Totaal bevat de library op het moment van testen 4 visualisaties. Alle functies die toegankelijk zijn voor de consumer van de library worden getest. Dit heeft als doel om de werking van de uiteindelijke output van de library te kunnen aantonen.

Om alle visualisaties te testen worden er voldoende test cases opgesteld, dat in ieder geval alle functionaliteit minimaal 1 keer uitgevoerd wordt. Het verwachtte gedrag wordt ingeschat o.b.v. de requirements en vergeleken met de uiteindelijke uitkomsten.

Om meer een hogere juistheid aan te tonen worden de testcases uitgevoerd in twee verschillende omgevingen, waarin bijna alle producten van ORTEC Sports draaien:

- AngularJS omgeving
- Angular 2+ omgeving

De acceptatiecriteria van een test is dan ook dat deze twee keer het verwachte gedrag moet vertonen in beide omgevingen.

2.2 Kwaliteit van de code

Naast het testen van de visualisaties in geheel, is er ook gekozen om elk onderdeel van de library individueel te testen. Dit wordt gedaan door alle publieke functies binnen de library te unit testen.

Om alle code individueel te testen, wordt er zoveel mogelijk code tijdens de ontwikkeling van de library getest. Dit wordt gedaan d.m.v het testing framework Jasmine. Aan het einde van iedere iteratie, waar wordt de geschreven code getest. Alle resultaten worden vastgelegd en wordt er d.m.v. een test coverage tool genaamd Istanbul reporter een test coverage rapport gegenereerd. Alle resultaten van de unit tests, zijn terug te vinden in hoofdstuk 4.

4. Testcases

ID	1
Beschrijving	Test de initialisatie van een veld
Requirement ID	5
Auteur	Wiechert Toonstra
Pre-conditie	Library is geïmporteerd in testproject
Teststappen	1. Creëer een van de 4 visualisaties 2. Set geen data als input 3. Visualiseer veld
Verwacht resultaat	Leeg veld zonder getekende objecten wordt gevisualiseerd.
Post-conditie	Veld object is gecreëerd.
Resultaat Angular 2+	Veld object is gecreëerd en wordt getoond in de browser.
Resultaat AngularJS	Veld object is gecreëerd en wordt getoond in de browser.
Fail/Pass	Pass
Aantekeningen	Create() aanroepen heeft voor de consumer geen enkele meerwaarde. Hierdoor kunnen namelijk de toegankelijke functies in verkeerde volgorde aangeroepen worden, wat er tot leidt dat visualisatie niet wordt getoond. Dit kan worden opgelost door dit naar constructor te verplaatsen.

ID	2
Beschrijving	Test de dimensies van een veld
Requirement ID	5
Auteur	Wiechert Toonstra
Pre-conditie	Library is geïmporteerd in testproject
Teststappen	1. Creëer een van de 4 visualisaties 2. Set geen data als input 3. Creëer een veld met de dimensies: 120px lang, 75px breed 4. Visualiseer veld
Verwacht resultaat	Leeg veld zonder getekende objecten wordt gevisualiseerd. Veld wordt in de browser ingeladen met dimensies van: 120px lang en 75px breed.
Post-conditie	Veld object is gecreëerd. SVG element is in browser volgens de juiste dimensies ingeladen.
Resultaat Angular 2+	Veld wordt ingeladen als SVG in de browser. SVG heeft afmetingen van 120px bij 75px.
Resultaat AngularJS	Veld wordt ingeladen als SVG in de browser. SVG heeft afmetingen van 120px bij 75px.
Fail/Pass	Pass
Aantekeningen	De visualisatie doet geen check op of de juiste dimensies voor een voetbalveld worden geconfigureerd. In theorie kan een voetbalveld met hele rare verhoudingen gevisualiseerd worden.

ID	3
Beschrijving	Test de initialisatie van een half veld
Requirement ID	6
Auteur	Wiechert Toonstra
Pre-conditie	Library is geïmporteerd in test project
Teststappen	1. Creëer een van de vier visualisaties 2. Kies half field als paramater 3. Set geen data als input 4. Pas veld dimensies lengte aan naar 60 5. Visualiseer veld
Verwacht resultaat	Leeg half veld wordt getekend.
Post-conditie	Veld object is geïnitaliseerd, zonder data
Resultaat Angular 2+	Leeg half veld met de afmetingen van 60x75 wordt getekend.
Resultaat AngularJS	Leeg half veld met de afmetingen van 60x75 wordt getekend.
Fail/Pass	Pass
Aantekeningen	Half veld moeten qua dimensies altijd wel worden aangepast. Schaalt niet automatisch mee met de keuze voor een half of heel veld, wat er heel raar uit kan zien.

ID	4
Beschrijving	Test de initialisatie van een Field Locations visualisatie met een blauwe driehoek die in het midden van het veld getekend wordt [50,50]
Requirement ID	17
Auteur	Wiechert Toonstra
Pre-conditie	Library is geïmporteerd in testproject
Teststappen	1. Creëer een field locations object 2. Creëer een field locations data object: blauwe driehoek met een coördinaat op [50,50] 3. Set data 4. Teken veld
Verwacht resultaat	Veld is getekend met een data punt die in het midden van het veld wordt getekend
Post-conditie	Field Locations object is gecreëerd
Resultaat Angular 2+	Veld wordt getekend met een blauwe driehoek in het midden.

Resultaat AngularJS	Veld wordt getekend met een blauwe driehoek in het midden.
Fail/Pass	Pass
Aantekeningen	

ID	5
Beschrijving	Test redraw-functionaliteit in de Field Locations visualisatie
Requirement ID	17
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd in testproject - Vorige test (3) is voltooid
Teststappen	1. Verander het coördinaat van het object dat gecreëerd is in testcase 3 naar [100,100], tijdens runtime 2. Herteken de visualisatie, tijdens runtime
Verwacht resultaat	Getekend object beweegt van [50,50] naar [100,100]
Post-conditie	Field Locations object heeft 1 getekend object, wiens coördinaat correct geschaald is
Resultaat Angular 2+	Blauwe driehoek verschijnt op [50,50]. Wanneer de coördinaten worden veranderd naar [100,100] en de visualisatie wordt hertekend, dan beweegt de driehoek naar [100,100]
Resultaat AngularJS	Blauwe driehoek verschijnt op [50,50]. Wanneer de coördinaten worden veranderd naar [100,100] en de visualisatie wordt hertekend, dan beweegt de driehoek naar [100,100]
Fail/Pass	Pass
Aantekeningen	

ID	6
Beschrijving	Test initialisatie van Grid Plot visualisatie, met een vakverdeling
Requirement ID	10
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd - Vakverdeling is gedefinieerd
Teststappen	1. Creëer grid objecten o.b.v. vakverdeling 2. Geef grid objecten aan Grid Plot visualisatie 3. Teken visualisatie
Verwacht resultaat	Een getekend veld die een grid bevat met de juiste dimensies en waardes.
Post-conditie	Grid visualisatie is getekend, met de juist geschaalde objecten.
Resultaat Angular 2+	Veld verschijnt met de een grid die volgens de juiste vakverdeling getekend is. Ook de waardes staan op de juiste plek en zijn correct.
Resultaat AngularJS	Veld verschijnt met de een grid die volgens de juiste vakverdeling getekend is. Ook de waardes staan op de juiste plek en zijn correct.
Fail/Pass	Pass
Aantekeningen	Foutgevoelig: als de vakverdeling niet juist wordt doorgegeven, worden vakken door elkaar heen getekend.

ID	7
Beschrijving	Test initialisatie en de start van Action Replay visualisatie
Requirement ID	19
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd
Teststappen	1. Creëer een Action Replay visualisatie 2. Creëer een lijst aan events, die in de Action Replay getoond kunnen worden. 3. Set deze lijst als data voor de Action Replay Visualisatie. 4. Start de replay
Verwacht resultaat	Veld wordt getekend. Op het moment dat de replay wordt gestart, verschijnen de events stuk voor stuk op het veld.
Post-conditie	
Resultaat Angular 2+	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm.
Resultaat AngularJS	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm.
Fail/Pass	Pass
Aantekeningen	

ID	8
Beschrijving	Test de pauze functionaliteit van de Action Replay visualisatie
Requirement ID	20
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd
Teststappen	7. Creëer een Action Replay visualisatie 8. Creëer een lijst aan events, die in de Action Replay getoond kunnen worden. 9. Set deze lijst als data voor de Action Replay Visualisatie 10. Start de replay 11. Pauzeer de replay bij event nr 5 12. Herstart de replay
Verwacht resultaat	Veld wordt getekend. Op het moment dat de replay wordt gestart, verschijnen de events stuk voor stuk op het veld. Als de Action Replay gepauzeerd wordt, dan blijft de replay gepauzeerd op het juiste event staan. Op het moment dat er weer start wordt gedrukt, gaat de Action Replay weer bij het juiste event verder.
Post-conditie	Timer is gepauzeerd bij het juiste element. Timer wordt hervat en gaat weer verder bij het juiste element.
Resultaat Angular 2+	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm. Wanneer er pauze wordt gedrukt, blijft de Action Replay bevroren totdat er weer start wordt gedrukt. Vervolgens hervat de Action Replay weer bij het juiste event.
Resultaat AngularJS	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm. Wanneer er pauze wordt gedrukt, blijft de Action Replay bevroren totdat er weer start wordt gedrukt. Vervolgens hervat de Action Replay weer bij het juiste event.
Fail/Pass	Pass
Aantekeningen	

ID	9
Beschrijving	Test initialisatie en de stop van Action Replay visualisatie
Requirement ID	20
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd
Teststappen	1. Creëer een Action Replay visualisatie. 2. Creëer een lijst aan events, die in de Action Replay getoond kunnen worden. 3. Set deze lijst als data voor de Action Replay Visualisatie. 4. Start de replay. 5. Stop de replay. 6. Start de replay opnieuw.
Verwacht resultaat	Veld wordt getekend. Op het moment dat de replay wordt gestart, verschijnen de events stuk voor stuk op het veld. Wanneer de Action Replay gestopt wordt, dan worden alle events van het veld weggehaald.
Post-conditie	
Resultaat Angular 2+	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm. Wanneer er stop wordt gedrukt, verdwijnen de getekende events. Wanneer er opnieuw gestart wordt, begint de replay opnieuw bij het eerste event.
Resultaat AngularJS	Events verschijnen stuk voor stuk met de vooraf ingestelde event-interval op het scherm. Wanneer er stop wordt gedrukt, verdwijnen de getekende events. Wanneer er opnieuw gestart wordt, begint de replay opnieuw bij het eerste event.
Fail/Pass	Pass
Aantekeningen	

ID	10
Beschrijving	Test initialisatie van de FieldLines visualisatie
Requirement ID	15
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd
Teststappen	1. Creëer een FieldLines visualisatie 2. Set geen data als input 3. Visualiseer veld
Verwacht resultaat	Veld wordt getekend. Omdat er geen data is geset, worden er geen lijnen getekend.
Post-conditie	Veld object zonder data is geinitaliseerd.
Resultaat Angular 2+	Veld object is gecreëerd en wordt getoond in de browser.
Resultaat AngularJS	Veld object is gecreëerd en wordt getoond in de browser.
Fail/Pass	Pass
Aantekeningen	

ID	11
Beschrijving	Test initialisatie en het tekenen van een lijn van [50,50] naar [100,100] van de FieldLines visualisatie
Requirement ID	15
Auteur	Wiechert Toonstra
Pre-conditie	- Library is geïmporteerd
Teststappen	1. Creëer een FieldLines visualisatie 2. Creëer een FieldLine object met een begin coördinaat [50,50] en een eind coördinaat [100,100] 3. Set FieldLine object als data 4. Visualiseer veld
Verwacht resultaat	Veld wordt getekend. Er verschijnt een lijn die loopt van de middenstip naar de hoekvlag rechtsonder in het veld.
Post-conditie	Veld object met data (1 lijn) is geinitialiseerd.
Resultaat Angular 2+	Veld wordt getoond, waarna er een lijn wordt getekend van de middenstip naar de hoekvlag rechtsonder

Resultaat AngularJS	Veld wordt getoond, waarna er een lijn wordt getekend van de middenstip naar de hoekvlag rechtsonder
Fail/Pass	Pass
Aantekeningen	

3 Resultaten unit tests

3.1 Unit tests

Test	Subject	Pass/Fail
should produce a full field	BaseVisuals	Pass
should produce a half field	BaseVisuals	Pass
should produce a penaltyBox	BaseVisuals	Pass
should produce a field as default	BaseVisuals	Pass
should create a fieldOverlay	BaseVisuals	Pass
should create a stand-alone full field	BaseVisuals	Pass
should create a stand-alone penaltyBox	BaseVisuals	Pass
should create a stand-alone half field	BaseVisuals	Pass
should draw a triangle	Drawables	Pass
should draw and redraw a line	Drawables	Pass
should filter drawables if in rectangle	Events	Pass
should filter fieldLines by startCoordinate, if in rectangle	Events	Pass
should filter fieldLines by startCoordinate, if in rectangle	Events	Pass
should create a circle factory	Factories	Pass
should create a square factory	Factories	Pass
should create a rectangle factory	Factories	Pass
should create a triangle factory	Factories	Pass
should create a line factory	Factories	Pass
should create a triangle	Factories	Pass
should create a square	Factories	Pass
should create a circle	Factories	Pass
should create a rectangle	Factories	Pass
should broadcast a mouse-click-event	Mouse Event Channel	Pass
should broadcast a mouse-out-event	Mouse Event Channel	Pass

should broadcast a mouse-over-event	Mouse Event Channel	Pass
should broadcast a drag-start-event	Mouse Event Channel	Pass
should broadcast a drag-move-event	Mouse Event Channel	Pass
should broadcast a drag-end-event	Mouse Event Channel	Pass
should create with container	SVG	Pass
should create with parent	SVG	Pass
should have a scaler	SVG	Pass
should scale x-coordinate properly	SVG	Pass
should scale y-coordinate properly	SVG	Pass
should create an action-replay visual	Visuals	Pass
should set action replay data with no errors	Visuals	Pass
should start action replay with no errors	Visuals	Pass
should pause action replay with no errors	Visuals	Pass
should stop action replay with no errors	Visuals	Pass
should create a field-grid visual	Visuals	Pass
should set grid-data with no errors	Visuals	Pass
should draw grid-data with no errors	Visuals	Pass
should redraw grid-data with no errors	Visuals	Pass
should create a field-lines visual	Visuals	Pass
should set field-line-data with no errors	Visuals	Pass
should draw fieldline data with no errors	Visuals	Pass
should draw and redraw fieldline data with no errors	Visuals	Pass
should create a field-locations visual	Visuals	Pass
should set fieldlocations data with no errors	Visuals	Pass
should draw fieldlocations data with no errors	Visuals	Pass

should draw and redraw fieldlocations data with no errors	Visuals	Pass

3.2 Total coverage

Definities coverage:

- Function coverage: aantal functies dat is uitgevoerd door de testen
- Statement coverage: aantal statements dat is uitgevoerd door de testen
- Branch coverage: aantal branches (mogelijke paden binnen een statement) dat is uitgevoerd door de testen
- Line coverage: aantal regels code dat is uitgevoerd door de testen

Total lines discovered	1579
Total lines coverage	1354 (85%)
Total statement coverage	1414/1643 (86.06%)
Total branches coverage	86/124 (69.35%)
Total functions coverage	305/363 (84.02%)

3.3 Global coverage

File		Statements		Branches		Functions		Lines	
spec		100%	338/338	100%	0/0	100%	90/90	100%	338/338
src/classes/constants		100%	49/49	100%	0/0	100%	0/0	100%	49/49
src/classes/drawables		86.27%	201/233	57.14%	24/42	82.61%	38/46	85.39%	187/219
src/classes/events		60.77%	79/130	72.73%	16/22	66.67%	26/39	59.2%	74/125
src/classes/factory-methods/shapes		69.51%	114/164	79.17%	19/24	80%	20/25	67.11%	102/152
src/classes/svg		82.8%	130/157	100%	6/6	81.13%	43/53	84.14%	122/145
src/classes/visuals		76.59%	193/252	37.5%	3/8	69.81%	37/53	75.62%	183/242
src/classes/visuals/base-visuals		96.82%	304/314	80%	16/20	89.29%	50/56	96.7%	293/303
src/classes/visuals/types		100%	6/6	100%	2/2	100%	1/1	100%	6/6

3.4 Coverage per type

3.4.1 Drawables

File		Statements		Branches		Functions		Lines	
arc.ts		95%	19/20	100%	0/0	75%	3/4	94.44%	17/18
circle.ts		96%	24/25	66.67%	4/6	100%	6/6	95.65%	22/23
field-line.ts		79.66%	47/59	66.67%	4/6	70%	7/10	78.95%	45/57
grid-item.ts		100%	35/35	100%	2/2	87.5%	7/8	100%	33/33
line.ts		93.1%	27/29	83.33%	10/12	100%	5/5	92.59%	25/27
rectangle.ts		80%	32/40	20%	2/10	85.71%	6/7	78.95%	30/38
triangle.ts		68%	17/25	33.33%	2/6	66.67%	4/6	65.22%	15/23

3.4.2 Events

File		Statements		Branches		Functions		Lines	
drag-filter.ts		100%	24/24	100%	14/14	100%	11/11	100%	23/23
drag-selector.ts		33.33%	25/75	0%	0/6	18.75%	3/16	31.51%	23/73
event-type.ts		100%	7/7	100%	2/2	100%	1/1	100%	7/7
mouse-events-channel.ts		95.83%	23/24	100%	0/0	100%	11/11	95.45%	21/22

3.4.3 Factories

File		Statements		Branches		Functions		Lines	
circle-factory.ts		64.29%	18/28	100%	2/2	75%	3/4	61.54%	16/26
line-factory.ts		66.67%	16/24	100%	2/2	75%	3/4	63.64%	14/22
rectangle-factory.ts		63.33%	19/30	100%	2/2	75%	3/4	60.71%	17/28
shape-factory-producer.ts		95.24%	20/21	64.29%	9/14	100%	3/3	95%	19/20
shape-factory.ts		100%	5/5	100%	0/0	100%	2/2	100%	4/4
square-factory.ts		63.33%	19/30	100%	2/2	75%	3/4	60.71%	17/28
triangle-factory.ts		65.38%	17/26	100%	2/2	75%	3/4	62.5%	15/24

3.4.4 SVG classes

File		Statements		Branches		Functions		Lines	
coordinate.ts		100%	5/5	100%	0/0	100%	2/2	100%	4/4
drawable-shape.ts		70%	35/50	100%	0/0	63.16%	12/19	73.91%	34/46
linear-scaler.ts		100%	19/19	100%	0/0	100%	6/6	100%	18/18
style-element.ts		100%	3/3	100%	0/0	100%	2/2	100%	2/2
svg-container.ts		84.42%	65/77	100%	6/6	86.96%	20/23	84.93%	62/73
svg-elem.ts		100%	3/3	100%	0/0	100%	1/1	100%	2/2

3.4.5 Visuals

File		Statements		Branches		Functions		Lines	
action-replay.ts		74.65%	53/71	100%	0/0	73.33%	11/15	73.53%	50/68
field-grid.ts		95.35%	41/43	50%	2/4	100%	8/8	95.12%	39/41
field-lines.ts		70.13%	54/77	25%	1/4	56.25%	9/16	69.33%	52/75
field-locations.ts		73.77%	45/61	100%	0/0	64.29%	9/14	72.41%	42/58

3.4.6 Base visuals

File		Statements		Branches		Functions		Lines	
base-field-visual-producer.ts		92.31%	12/13	66.67%	4/6	50%	2/4	91.67%	11/12
field-overlay.ts		95.74%	45/47	0%	0/2	83.33%	10/12	95.65%	44/46
field-visual-type.ts		100%	4/4	100%	2/2	100%	1/1	100%	4/4
field-visual.ts		93.33%	42/45	100%	2/2	87.5%	7/8	93.02%	40/43
field.ts		95.12%	39/41	100%	2/2	87.5%	7/8	94.87%	37/39
half-field.ts		99.16%	118/119	100%	4/4	100%	15/15	99.14%	115/116
penaltyBox.ts		97.78%	44/45	100%	2/2	100%	8/8	97.67%	42/43

3.4.7 Constants

File		Statements		Branches		Functions		Lines	
field-constants.ts		100%	28/28	100%	0/0	100%	0/0	100%	28/28
half-field-constants.ts		100%	12/12	100%	0/0	100%	0/0	100%	12/12
penalty-box-constants.ts		100%	9/9	100%	0/0	100%	0/0	100%	9/9