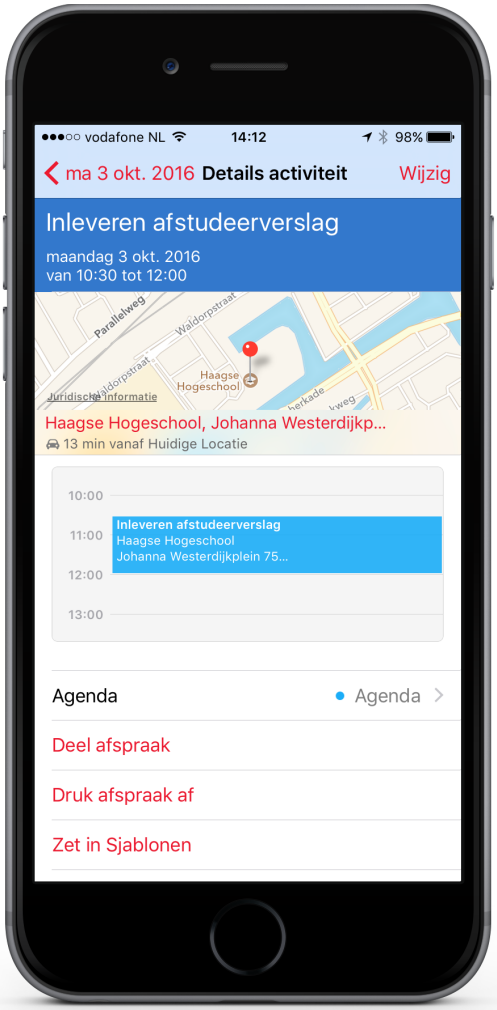
DE HAAGSE HOGESCHOOL

Week Calendar

Afstudeerverslag

WeekCal Framework



Auteur Guus Mulder

Datum 3 Oktober 2016

Afstudeerverslag

WeekCal

Week Calendar

Auteur: Guus Mulder
Studentnummer: 12011029
School: De Haagse Hogeschool
Opleiding: Informatica
Eerste Examinator: A.J. Toet
Tweede Examinator: H.J. Middendorp
Bedrijf: WeekCal
Locatie: Wateringen
Datum: 3 Oktober 2016
Periode: Mei 2016 - Oktober 2016

Referaat

- iOS
- Swift
- Objective-C
- iPhone
- iPad
- Native
- Mobiele applicatie
- Webbeat
- WeekCal
- Week Calendar
- Framework
- Converteren

Voorwoord

Mijn naam is Guus Mulder, ik ben 21 jaar en studeer informatica aan de Haagse Hoge School. Ik heb het afgelopen half jaar een fijne en uitdagende afstudeer periode uitgevoerd en heb deze afgesloten met het tot stand komen van dit verslag.
Mijn dank gaat uit Saurabh Garg en Geert Merkelbach van
WeekCal te Wateringen en de heer A.J. Toet van de Haagse Hoge School.
Zonder hun flexibiliteit en hulp had deze afstudeer opdracht niet in deze prettige vorm tot stand kunnen komen.

Ik hoop met dit verslag een bijdrage te kunnen leveren tot vergroting van de kennis tot deze materie en daarmee ook andere studenten en ontwikkelaars enthousiast te maken voor het bouwen van mobiele applicaties.

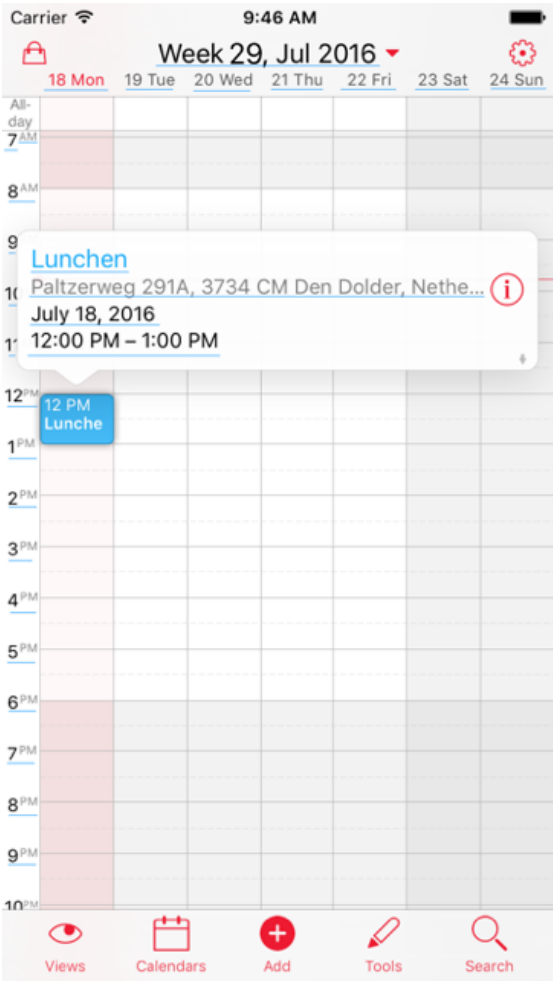
Inhoudsopgave

1. Inleiding

Het afgelopen half jaar heb ik afgestudeerd bij WeekCal, een ICT en marketing bedrijf gevestigd in Wateringen.
WeekCal is een bedrijf dat zich richt op het ontwikkelen en publiceren van mobiele applicaties. Een van de grootste en meest succesvolle mobiele applicaties van WeekCal is de applicatie 'Week Calendar', een agenda applicatie voor iPhone en iPad. De Week Calendar app is echter niet één app maar twee, één voor iPhone en één voor iPad. Beide applicaties zijn bijna identiek, dit betekend dubbele code.
Dit is waar ik in het plaatje kwam, namelijk als ontwikkelaar van een gezamenlijk framework voor de iPhone en iPad applicatie, oftewel mijn afstudeer opdracht. Het doel van het 'WeekCal Framework' was code centraliseren zodat deze kon worden hergebruikt en dit doel is ook zeker bereikt.

Gedurende mijn afstudeer periode heb ik het WeekCal Framework ontworpen, gebouwd, getest en gedocumenteerd. Aan het einde van mijn afstudeer periode had ik een net, herbruikbaar en volledig testbaar framework afgeleverd dat binnenkort in productie gaat.

In afbeelding 1 staat de home pagina van de Week Calendar app afgebeeld. Alles dat is onderstreept met een blauwe streep wordt vanuit het WeekCal Framework gegenereerd. In dit plaatje dus: Teksten, datum, tijden, evenementen, vertalingen, tijdzones, locatiebepaling en evenement kleur. Ook de thema kleuren worden bepaald door het framework.
Wat al deze functies inhouden en hoe het framework tot stand is gekomen kom ik later op terug in dit verslag.



Figuur 1. Framework in Week Calendar

2. WeekCal

In dit hoofdstuk wordt beschreven wat WeekCal is, hoe het te werk gaat en op welke wijze ik eraan mee werk.

2.1 WeekCal

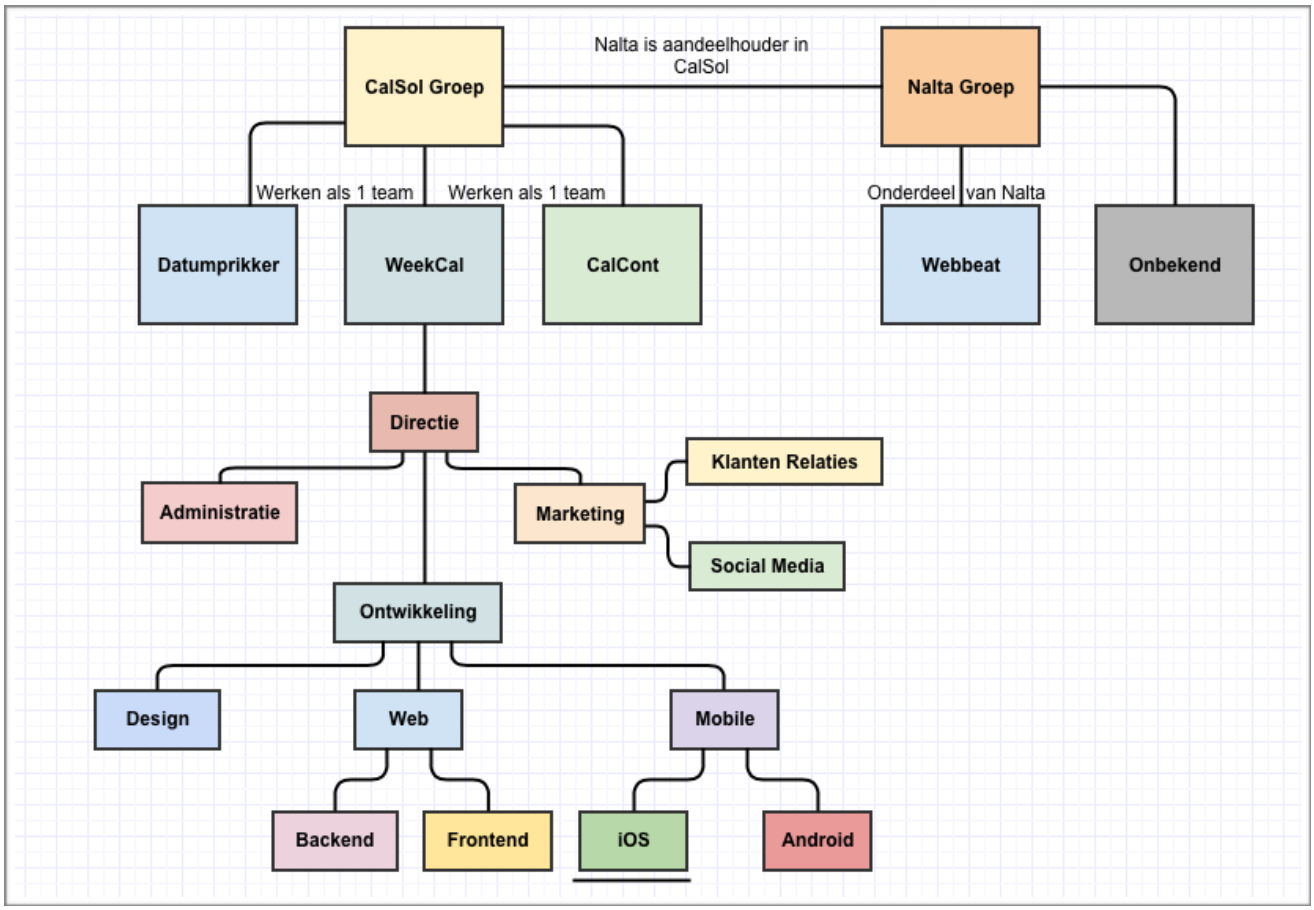
WeekCal, gevestigd in Wateringen, is een ICT bedrijf dat gespecialiseerd is in het ontwikkelen, onderhouden, publiceren en promoten van mobiele applicaties. Deze vier specialiteiten werken zich uit tot één doel, een product van hoge kwaliteit afleveren dat veel gebruikers trekt. Een van de grootste en meest succesvolle mobiele applicaties van WeekCal is de applicatie 'Week Calendar'. Week Calendar is een agenda applicatie die momenteel op nummer 2 staat in de Apple app store onder de categorie Productiviteit (betaald). De Week Calendar app is dan ook de applicatie waar WeekCal zich het meest op richt. De Week Calendar app heeft wereldwijd ruim een miljoen zeer actieve gebruikers.

2.2 WeekCal Doelgroep

De primaire doelgroep van WeekCal bestaat uit smartphone eigenaars die niet tevreden zijn met de standaard agenda in iOS en Android. Dit is de doelgroep die het meeste baat heeft bij een mobiele applicatie als Week Calender, een app die niet gratis is. WeekCal ontwikkelt geen applicaties voor andere bedrijven en ontwikkelt dus alleen apps voor zichzelf, de doelgroep bestaat daarom uit app gebruikers en niet uit bedrijven die apps nodig hebben. Na de primaire doelgroep is er ook een grotere doelgroep die bestaat uit alle smartphone eigenaars, de primaire doelgroep is hier dus een specifiek deel van. Alle smartphone eigenaars zijn potentiële gebruikers van de Week Calendar app maar sommige type smartphone eigenaars zullen niet snel de app aanschaffen, hieronder vallen: Kinderen tot 12, gebruikers zonder betaalmethode en ouderen.

2.3 Organisatie Structuur

WeekCal is een klein bedrijf, het bestaat uit één pand met twee verdiepingen waar alle werknemers elkaar regelmatig tegenkomen. Niet alleen het personeel, maar ook de directeuren werken in hetzelfde pand. Hierdoor is het mogelijk om snel en ongehinderd contact met elkaar te hebben, wat leidt tot snellere en betere communicatie en daardoor vaak ook beter uitgevoerde projecten. Het is dus duidelijk dat WeekCal een platte en informele organisatiestructuur heeft. WeekCal is echter onderdeel van een grotere organisatie met een complexe structuur, in het organogram hieronder (volgende pagina) wordt deze structuur afgebeeld.



Figuur 2. Organogram WeekCal

Zoals te zien is in figuur 2 is de structuur van WeekCal niet zo gemakkelijk als dat het in eerste instantie leek. Hieronder volgt een beschrijving van alle elementen boven WeekCal in het organogram.

CalSol Groep: Dit is een overkoepelend bedrijf voor Datumprikker, WeekCal en CalCont. De drie sub-bedrijven werken samen als 1 team.

Nalta Groep: Groep van bedrijven waaronder Webbeat. Nalta is een aandeelhouder in de CalSol groep.

Webbeat: Het bedrijf dat de producten van CalSol ontwikkeld. Webbeat is het bedrijf dat in hetzelfde pand opereert als WeekCal en Datumprikker maar dan op de 2e verdieping. WeekCal & Datumprikker zitten op de begane grond.

Hoewel er dus een complexe structuur aanwezig is, wordt er in dit verslag alleen gefocust op WeekCal en Webbeat. Dit komt doordat de bovenstaande structuur geen effect heeft gehad op mijn afstudeer opdracht, er was alleen contact met werknemers van WeekCal en Webbeat.

2.3.1 Beschrijving functies en afdelingen WeekCal

Functies

Directeur
De directeur van Weekcal werkt in de afdelingen administratie en klanten relaties.

Marketeer
Marketeers werken in de afdeling marketing. Zij zorgen ervoor dat het gezicht van WeekCal er goed uit ziet. Hiernaast is het hun doel om nieuwe opdrachten binnen te halen.
Software Engineer
Software engineers werken binnen de afdelingen Web en Mobile. De software engineers ontwikkelen alle web -en mobiele applicaties.
Designer
Designers werken binnen de afdeling design. De taak van de designer is het ontwerpen van user interfaces, icons en andere visuele assets.
Afstudeer/Stage Mentor
Een afstudeer/stage mentor is verantwoordelijk voor bepaalde afstudeerders/ stagiairs. Afstudeer/Stage mentoren kunnen theoretisch gezien zich bevinden in iedere afdeling, met als voorwaarde dat hun stagiairs zich in diezelfde afdelingen bevinden.

Afdelingen

Administratie
Hier vallen de financiën en HRM onder.
Marketing

Klanten Relaties
De afdelingen klanten relaties verzorgt opdrachten en houd relaties met al bestaande klanten gezond.
Social Media
De social media afdeling regelt voor alle producten de Facebook, Twitter en andere social media accounts. Inclusief blogs en vlogs.

Ontwikkeling

De ontwikkeling afdeling bestaat uit drie onderdelen, namelijk Design, Web en Mobile. Binnen deze afdelingen worden producten ontwikkeld van design naar gerealiseerd product.
Design
De design afdeling verzorgt user interfaces, art en vergelijkbare visuele assets.
Web
De afdeling web is verantwoordelijk voor het ontwikkelen van websites en web applicaties.

Backend
De backend afdeling verzorgt database's, API's en andere web services. Dit staat ook in connectie voor Mobiele applicaties die moeten communiceren met een API.
Frontend
De frontend afdeling zet user interface designs om in werkende interfaces. Dit wordt met html, css en javascript gedaan.

Mobile
De mobile afdeling ontwikkelt mobiele applicaties voor iOS en Android, deze zijn echter wel gescheiden van elkaar.

iOS
De iOS afdeling maakt mobiele applicaties voor iOS met XCode. Dit is de afdeling waarin ik mijn afstudeer opdracht heb uitgevoerd.
Android
De Android afdeling maakt mobiele applicaties voor Android met Xamarin.

2.3.2 Afstudeer begeleiding

WeekCal legt groot belang bij het begeleiden van haar afstudeerders. Iedere afstudeerder heeft de gehele afstudeer periode lang één afstudeer mentor. De afstudeer mentor staat altijd klaar om te helpen met problemen en op die manier de afstudeer opdracht in goede banen te leiden. Met de afstudeer mentor wordt naast een wekelijkse scrum review ook iedere twee weken een voortgangsgesprek gehouden. Tijdens dit voortgangsgesprek wordt besproken hoe het met de afstudeer opdracht gaat en wat de huidige situatie van de afstudeerder binnen het bedrijf is. Tot slot wordt er ook iedere maand een evaluatie gesprek gehouden met de afstudeer mentor. Tijdens het evaluatie gesprek wordt besproken wat er goed gaat en wat er beter kan.

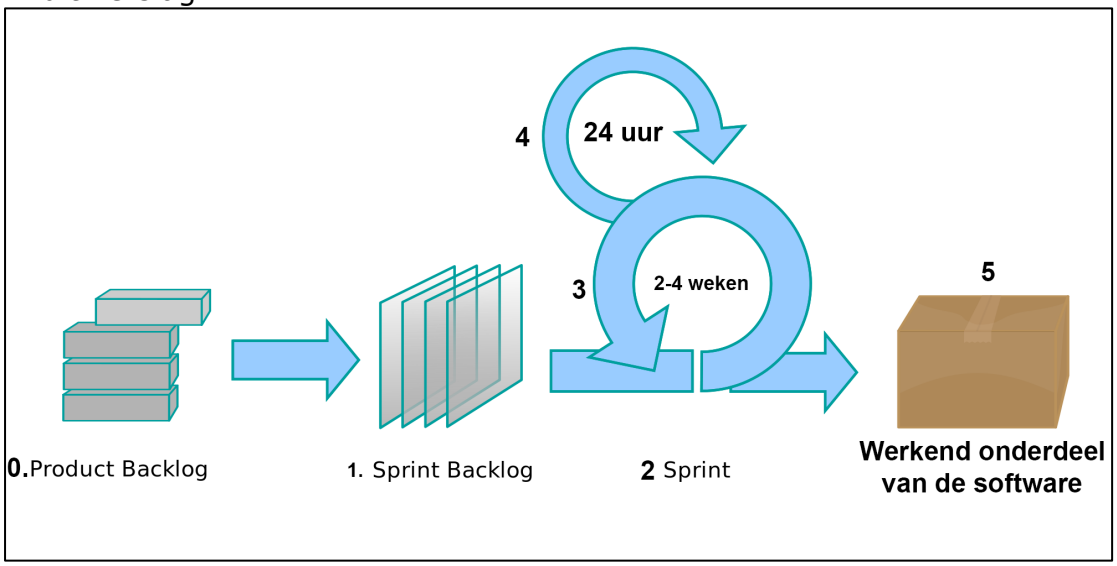
2.4 Mijn werkomgeving

Voor mijn afstudeer opdracht was er geen protocol opgesteld waaraan ik mij moest houden. Zonder protocol werken is lastig en om die reden heb ik zelf in samenwerking met mijn afstudeer mentor een protocol opgesteld. In de volgende sub hoofdstukken wordt beschreven waaruit dit protocol bestond.

2.4.1 Methodes en Technieken

Project management methode

De project management methode is de manier waarop het project wordt gepland en uitgevoerd. Bij WeekCal is er geen standaard project ontwikkeling methode. Tijdens het project heb ik gekozen om met scrum te werken, de motivatie hiervoor volgt later in dit verslag.



Figuur 3. Scrum cyclus uitgebeeld, bron: literatuurlijst: 1.

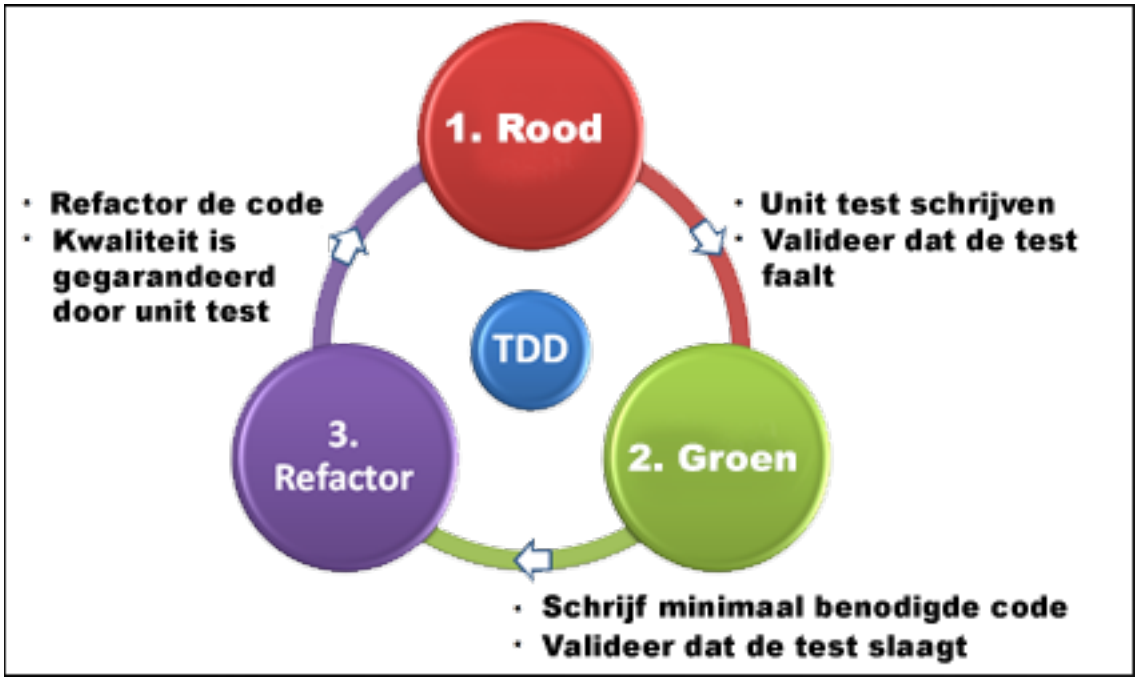
Scrum is een Agile aanpak, welke als framework kan worden ingezet om software in teamverband te ontwikkelen. De Scrum methode is iteratief; zo wordt er gewerkt in korte iteraties (Figuur 3, punt 3) van twee tot maximaal vier werken, die ook wel 'Sprints' (Figuur 3, punt 2) worden genoemd. Op iedere ochtend van de sprint (Figuur 3, punt 4) wordt er een stand-up meeting gedaan waarin wordt gesproken over de sprint en haar progressie. Iedere sprint wordt afgerond met een werkend onderdeel van de ontwikkelen software (Figuur 3, punt 5). Het werkende onderdeel bestaat uit de sprint backlog (Figuur 3, punt 1), dit zijn een aantal user stories die samen een

werkend stuk software vormen. Deze user stories worden gehaald uit de product backlog (Figuur 3, punt 0), dit is een verzameling van alle requirements van het project. Het product backlog kan tijdens het project worden aangepast waardoor scrum nog flexibeler wordt.

Een groot verschil tussen Scrum en andere methodes is dat bij scrum al het werk (ontwerpen, bouwen en testen) telkens binnen één sprint wordt uitgevoerd. Hiernaast geldt bij scrum het principe 'iedereen doet alles', wat betekent dat ieder lid van het scrum team alle taken uitvoert (ontwerpen, bouwen en testen). Dit principe werkt goed wanneer het team uit slechts een persoon bestaat, omdat iedereen dan toch alles doet en de methode hierop aansluit.

Systeemontwikkeling methode

De systeemontwikkeling methode is de manier waarop software wordt ontwikkeld, vaak bestaande uit principes en regels. WeekCal heeft in de huidige situatie geen standaard systeemontwikkeling methode, om die reden heb ik zelf een goed passende methode moeten kiezen. Ik heb uiteindelijk gekozen voor de systeemontwikkeling methode Test-Driven Development.



Figuur 4. Test Driven Development

De regel bij test-driven development (TDD) is dat voor functies eerst een test wordt geschreven en daarna pas de implementatie; in plaats van andersom. Hierdoor wordt er verzekerd dat:

- De verwachte output van tests correct is
- Alle functies testbaar zijn
- Fouten snel worden ontdekt

Deze eigenschappen maken TDD een goede keuze voor het ontwikkelen van een framework dat code van hoge kwaliteit nodig heeft.

2.4.2 Gebruikte tools

Integrated development environment
Tijdens mijn afstudeer periode heb ik twee IDE's (Integrated Development Environment) gebruikt. Een IDE is een tool die het de software engineer gemakkelijker maakt om code te maken. Het kiezen van een goede IDE is van groot belang voor de uitkomst van een project, de IDE's die ik gebruikt heb waren XCode en Eclipse.

XCode
XCode is een tool die door Apple wordt uitgegeven om IOS applicaties te bouwen. De tool is zeer geavanceerd en kost niets. Naast applicaties bouwen heeft Xcode ook een geïntegreerde test suite, namelijk XCTest. Hiermee kunnen geautomatiseerde tests worden gemaakt.

Eclipse
Eclipse is een tool die gebruikt word om java code te schrijven, de tool is gratis en open source. Ik heb de tool meerere keren gebruikt bij het schrijven van scripts voor de afstudeer opdracht.

Visual Paradigm
Voor het ontwikkelen van klasse diagrammen heb ik gebruik gemaakt van de tool Visual Paradigm. Deze tool heb ik ook vaak tijdens mijn studie gebruikt, wat dus goed uitkwam.

2.4.3 Gebruikte talen

Swift / Objective-C
Dit is de programmeertaal die wordt gebruikt in de tool XCode. Swift is de opvolger van Objective-C maar werkt nog steeds in combinatie met zijn voorganger. Zo kan er gebridged worden tussen Swift en Objective-C, wat betekent dat modules die zijn geschreven in Objective-C ook kunnen worden gebruikt in programma's die in Swift zijn geschreven. Het WeekCal framework is geschreven in Swift, de WeekCall app is echter nog geschreven in Objective-C. Dit betekent dat het WeekCal framework volledig gebridged moet kunnen worden naar Objective-C

Java
Java is de programmeertaal die wordt gebruikt in de tool Eclipse. Tijdens mijn afstudeer periode heb ik meerdere scripts geschreven voor automatisering in Java.

JSON
JSON is een taal die wordt gebruikt voor het opslaan en ophalen van gegevens. Tijdens mijn afstudeer opdracht heb ik gebruik gemaakt van JSON voor het ophalen van onder anderen evenementen.

2.4.4 Richtlijnen testen en productie

Betrouwbaarheid product
Voordat ontwikkelde producten in productie kunnen worden gebracht, moeten deze eerst intensief worden getest. WeekCal heeft een zeer grote en kritische user base, er mogen dus absoluut geen grote problemen in productie komen. In de huidige situatie heeft WeekCal voor iedere release een beta ronde van circa 2 weken, oftewel, een gebruikers test. De implementatie van het WeekCal framework is echter zo'n grote

verandering dat alleen een beta periode niet genoeg is, hier was helaas geen protocol voor bij WeekCal. Om deze reden heb ik zelf een oplossing moeten bedenken, namelijk het gebruik van Test-Driven Development, wat de kwaliteit van code garandeert en daarmee dus de betrouwbaarheid van het framework versterkt.

In productie brengen
Zoals eerder beschreven wordt voordat een product in productie wordt gebracht er eerst een minimaal twee weken lang durende beta periode gestart. Indien de beta periode positieve resultaten oplevert, wordt het product in productie gebracht. Indien de beta periode problemen aantoont, moet indien deze problemen kritiek zijn het product worden aangepast, opnieuw in beta worden gezet en de beta periode worden verlengd met minstens een week. Deze cyclus gaat door totdat er expliciet wordt gekozen om toch in productie te gaan of als er geen problemen meer voorkomen. Na het in productie brengen van een product wordt er intensief op fouten gelet zodat er bij problemen snel kan worden gereageerd met oplossingen.

3. Opdracht Omschrijving

In dit hoofdstuk is beschreven wat mijn afstudeer opdracht in heeft gehouden en welke doelen er bereikt moesten worden.

WeekCal heeft op dit moment twee verschillende Week Calendar apps in de App Store, voor zowel iPhone als iPad is er namelijk een aparte iOS applicatie ontwikkeld. Beide apps gebruiken zo veel mogelijk dezelfde codebase, beide met specifieke branches in de user interface elementen. De iPhone applicatie heeft meer features en functionaliteit dan zijn iPad tegenhanger. Ook al gebruiken deze applicaties voor een groot deel dezelfde architectuur, toch worden tijdens het bouw proces beide projecten als individuele codebases behandeld. Het effect hiervan is dat veranderingen die tijdens het ontwikkelen van één van de apps zijn gemaakt niet automatisch mee veranderen op de andere app.
De huidige oplossing in het delen van de codebase zorgt voor de volgende problemen:
- Gelijktijdige ontwikkeling aan de beide apps is complex (of onmogelijk);
- Het aantal ontwikkelaars wat gelijktijdig aan de apps kan werken wordt beperkt door de snel toenemende complexiteit bij het doorvoeren van aanpassingen in de codebase;
- Nieuwe ontwikkelingen kunnen minder eenvoudig op de achtergrond als beta-feature meegenomen wordt in een normale release;
- Code kan niet hergebruikt worden in andere apps zoals bijvoorbeeld die van Datumprikker.

Het streven van dit project is het ontwikkelen van een WeekCal framework dat alle functionaliteit aanbiedt van de al bestaande iPhone en iPad Week Calendar iOS app, m.u.v. de user interfaces. Vervolgens kunnen de iPhone en iPad applicatie beiden het framework gebruiken en dus, op user interface na, hun codebase weghalen. Wanneer er een update/verandering is binnen het framework kunnen beide applicaties direct worden geüpdatet met de nieuwste codebase, in plaats van dat beide applicaties hun codebase moeten aanpassen. Het doel dat hiermee bereikt wordt is het vergemakkelijken van het onderhoud van de iPhone en iPad applicatie. Doordat er slechts één gescheiden codebase als framework is, is consistentie geen probleem meer, kunnen meerdere ontwikkelaars tegelijk eraan werken en kan code gemakkelijk worden hergebruikt door andere applicaties die het framework gebruiken. Door deze verandering kost het onderhouden van de Week Calendar applicaties minder tijd en geld. Tijdens het ontwikkelen van het WeekCal framework is het de bedoeling dat de functionaliteit die in het framework worden geplaatst ook direct naar Swift code worden omgezet en wanneer deze functionaliteit van oude API's gebruik maken, deze moderniseren.

3.1 Doelen

Het WeekCal framework heeft een aantal hoofddoelen, namelijk:

- Het vergemakkelijken van updates van de iPad en iPhone week calendar applicatie
- Verzekeren dat iPhone en iPad dezelfde functionaliteit hebben
- Verouderde code moderniseren

Vergemakkelijken van updates
Het primaire doel van het WeekCal framework is het vergemakkelijken van updates van de iPhone en iPad applicatie. In de huidige situatie moet bij een update van de Week Calendar applicatie de bron code van zowel de iPhone als de iPad app worden bijgewerkt. Het veranderen van broncode kan altijd tot problemen en inconsistenties leiden wat leidt tot extra ontwikkel tijd en meer risico's. De oplossing die hiervoor is bedacht is het WeekCal framework, door in het framework code te centraliseren en herbruikbaar te maken kunnen de iPhone en iPad applicatie gebruik maken van hetzelfde framework. Wanneer bij updates er dan functionaliteit wordt aangepast kan dit in het framework worden gedaan en hoeven de iPhone en iPad applicatie niet

veranderd te worden. Om dit doel te bereiken moet het framework bruikbaar zijn voor zowel iPhone als iPad en dus generiek zijn, dit is een van de hoofdprincipes die later in het verslag wordt uitgewerkt.

Verzekeren dat iPhone en iPad dezelfde functionaliteit hebben
In de huidige situatie zijn er veel inconsistenties in de code base van de iPhone en iPad applicatie. Hoewel dit voor beide applicaties individueel niet uitmaakt, zijn deze inconsistenties een vervelend probleem bij het bouwen van dezelfde functionaliteit in zowel de iPhone als iPad applicatie. Er kan namelijk niet een gezamenlijke code gebruikt worden door onder anderen verschillende namen en functies. Door verdere functionaliteit in het WeekCal framework op te nemen en al bestaande functionaliteit te porteren naar het framework kan er verzekerd worden dat zowel de iPhone als iPad applicatie functionaliteit hebben die exact hetzelfde gedrag heeft.

Verouderde code moderniseren
Het laatste doel van het WeekCal framework is het moderniseren van verouderde code, dit houdt in dat code die:

- Met oude syntax is geschreven
- Gebruik maakt van oude API's
- Niet meer ondersteund wordt

wordt herschreven naar code die dezelfde functionaliteit biedt en moderne methodes en syntax gebruikt. Verouderde code wordt vaak niet meer ondersteund door iOS en wordt ook niet meer geüpdatet, hiernaast hebben nieuwe methodes vaak ook performance of bruikbaarheid verbeteringen. In een normale productie situatie is er vaak geen tijd om code te moderniseren, maar bij het ontwikkelen van het nieuwe framework, waarbij veel code herschreven moet worden, komt het juist goed uit.

3.2 Uitgangssituatie

3.2.1 Resources
Mijn afstudeeropdracht was een compleet nieuw project waar nog geen start aan was gemaakt. Dit betekent dat er nog geen requirements, design, code, documentatie of tests voor waren ontwikkeld. De afstudeeropdracht bestond echter uit zowel het ontwikkelen van compleet nieuwe modules als het omzetten van Objective-C code uit de Week Calendar app naar het WeekCal framework in Swift. Van de Week Calendar app was alleen de code voor mij beschikbaar, de code was niet gedocumenteerd, er waren geen tests, geen requirements en ook geen systeem design diagram. Dit maakte het lastig om snel bekend te worden met de code base van de Week Calendar app, dit is ook weer een van de redenen dat ik heb gekozen om met TDD te werken. Werken met een onbekende code base levert namelijk altijd problemen, test-driven development zorgt ervoor dat fouten in de code direct duidelijk worden door het constant maken van unit-tests.

3.2.2 Protocollen, methodes en technieken
Voor het WeekCal framework project was geen systeem ontwikkelingsmethode en ook geen project management methode gekozen. WeekCal had ook geen standaard protocol waaraan ik mij moest houden, ik heb zelf zowel de systeem ontwikkelingsmethode als project management methode moeten kiezen.

4. Aanpak en uitvoering

Tijdens mijn afstudeer periode heb ik zoals eerder beschreven in de opdracht omschrijving gewerkt aan het WeekCal framework project. Het gehele project heb ik uitgevoerd met de projectmanagement methode scrum en de systeemontwikkeling methode test-driven development. Binnen scrum sprints heb ik telkens een stukje werkende en geteste code gebouwd binnen het framework, deze stukjes code werden dan modules genoemd. Het WeekCal framework project bestond in grote lijnen uit 3 fases, namelijk:

- Initiatie
- Framework ontwikkeling
- Framework implementatie

Deze fases zijn zoals hierboven beschreven in chronologische volgorde uitgevoerd, iedere fase bestond uit een collectie taken waaruit een of meerdere producten zijn gekomen.

Fase	Percentage Tijd	Taken	Producten
Initiatie	15%	Plannen, methodes kiezen, requirements analyseren	- Plan van Aanpak - Requirements Rapport - Master test plan
Framework ontwikkeling	55%	Ontwerpen, Programmeren en Testen van WeekCal Framework Schrijven documentatie van WeekCal Framework	- System Design Diagram - Test rapportages - Deprecation modernisation rapport - WeekCal Framework - Documentatie WeekCal Framework
Framework implementatie	30%	WeekCal Framework bijschaven wanneer nodig (ontwerpen, programmeren, testen WeekCal Framework). Programmeren en testen van WeekCal implementatie in Week Calendar app	- Week Calendar app werkend met WeekCal framework implementatie - Test rapportages

Tabel 1. Project Fases

In tabel 1 is weergegeven welke taken er worden uitgevoerd in welke fase en welke producten hieruit zijn voortgekomen. Binnen dit hoofdstuk worden de fases, taken en producten verder uitgewerkt

4.1 Initiatie

De initiatie fase is de eerste fase van het WeekCal project, het doel van deze fase was het maken van een goede start van het WeekCal framework project. Om dit doel te bereiken moest in deze fase er een duidelijk beeld komen van het project, bestaande uit een requirements analyse en plan van aanpak. Zoals eerder beschreven in hoofdstuk 3.2 was er nog geen standaard project ontwikkeling -en management methode, geen requirements, geen planning en ook geen codebase.

Tijdens de initiatie fase moest ik hier verandering in brengen. Aan het begin van de initiatie fase heb ik een plan van aanpak geschreven waarin de opdracht, planning en het beoogde resultaat staat beschreven. Vervolgens heb ik op basis van het plan van aanpak een requirements analyse uitgevoerd met als resultaat een requirements rapport. Aan de hand van het requirements rapport was het mogelijk om met overtuiging de correcte project management -en ontwikkeling methode te kiezen en hiermee de initiatie fase af te sluiten.

4.1.1 Requirements Rapport

Bij de opstart van het WeekCal framework luidde de opdracht 'ontwikkelen van WeekCal framework met alle functionaliteit van de Week Calendar app'. Dit is een goed streven maar is veel te abstract om naartoe te werken. Om deze reden moest dit doel worden omgezet naar een collectie requirements. Tijdens het opstellen van de requirements waren er twee hoofdvragen:

- Over welke functionaliteit beschikt de Week Calendar app
- Welke functionaliteit wil WeekCal in het framework hebben

Beide vragen moesten eerst worden beantwoord voordat ik de requirements kon opstellen, zonder ophelderingen over deze onderwerpen zou ik een onvolledig beeld hebben van zowel de Week Calendar app als het WeekCal framework. In overleg met mijn afstudeer mentor, die behalve afstudeer mentor ook de lead iOS developer van WeekCal is, heb ik beide vragen kunnen beantwoorden.

Functionaliteit Week Calendar app globaal

nr	Functionaliteit
1	Events plannen
2	Contacten ophalen en uitkiezen
3	Agenda's ophalen en aanpassen/verwijderen/toevoegen
4	Agenda's kopen
5	Evenementen slepen (tijd veranderen)
6	Evenementen zoeken
7	Kleuren aanmaken
8	Evenementen afdrukken
9	Kleuren schema's
10	Agenda instellingen
11	App beoordelingen
12	Dropbox integratie
13	Lijst, Dag, Agenda, Week, Mini Maand, Maand en Jaar weergave

nr	Functionaliteit
14	Dagen kiezen
15	Evenementen delen (airdrop, mail, sms, whatsapp, Due, appigo Todo, Things, OmniFocus)
16	Notificaties van verjaardagen en evenementen
17	Popups, popovers, alerts
18	Geocoding

Tabel 2. Globale functionaliteit Week Calendar App

In tabel 2 staat globaal beschreven over welke functionaliteit de Week Calendar app beschikt, voor ieder nummer in de tabel bestaat de functionaliteit uit meerdere kleine functies en eventuele interfaces. Met gebruik van de bovenstaande tabel kon ik samen met mijn afstudeer mentor bespreken welke functionaliteit in het framework moest worden gebouwd.

Een van de hoofdeisen van het WeekCal framework is: 'Het WeekCal framework moet generiek zijn', dit betekent dat het framework niet specifiek voor één target (device/ app/project) moet worden gemaakt maar juist bruikbaar moet zijn voor verschillende targets. Dit betekend dat functionaliteit die specifiek is voor bijvoorbeeld iPad in eerste instantie niet in het framework gebouwd moet worden, er zijn uitzonderlijke gevallen waarbij target specifieke functionaliteit ook beschikbaar moet worden voor andere targets. In dat geval kan de target specifieke functionaliteit wel in het framework worden gebouwd.

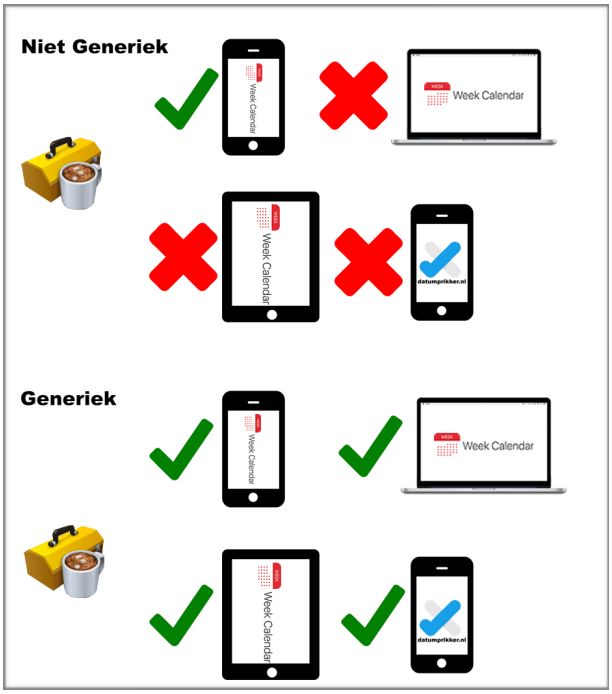
Één van de meest niet-generieke functionaliteit is user-interfaces, deze worden dus ook niet in het framework gebouwd. Een klein aantal uitzonderingen op deze regel worden later in dit verslag beschreven.

Waarom Generiek?

Het doel van het WeekCal framework is onder anderen om de codebase van de Week Calendar app herbruikbaar te maken voor zowel de iOS als iPad applicatie. Om dit doel te bereiken moet er ten alle tijden generiek gewerkt worden. Target specifieke code zal namelijk het framework incompatibel maken met andere targets. Hiernaast heeft een generiek framework nog meer voordelen, namelijk:

- Bruikbaar voor andere projecten zoals datumprikker.nl en plan2meet
- Ondersteunt alle Apple apparaten (iPhone, iPad, iPod, Apple Watch, Apple TV)
- Over het algemeen complexere, maar wel beter ontworpen code

Behalve voordelen heeft generiek werken ook zijn nadelen, het is namelijk lastig om code generiek te houden. Nette designs worden vaak lelijk en te complex door het constant bouwen van nieuwe functionaliteit binnen de code, uiteindelijk is er dan niets meer over van



Figuur 5. Niet generiek VS generiek

het initiële design en is de code niet meer generiek.

De manier waarop bij het WeekCal hier rekening mee wordt gehouden is door expliciet geen niet-generieke functionaliteit te bouwen in het framework. Dit is natuurlijk geen verzekering voor de toekomst van het framework, maar zorgt er wel voor dat het framework aan het einde van mijn afstudeer periode generiek is.

Minimale Functionaliteit WeekCal framework globaal

nr	Functionaliteit
1	Events Plannen
2	Contacten ophalen en uitlezen
3	Agenda's ophalen, wijzigen, verwijderen en toevoegen
4	Agenda's kopen
5	Kleuren aanmaken
6	Kleuren schema's
7	Geocoding
8	Agenda instellingen
9	Notificaties
10	Popups, popovers en alerts creatie
11	Geocoding
	Afgevallen functionaliteit
1	Agenda's ophalen en aanpassen/verwijderen/toevoegen
2	Evenementen slepen (tijd veranderen)
3	Evenementen zoeken
4	Evenementen afdrukken
5	App beoordelingen
6	Dropbox integratie
7	Lijst, Dag, Agenda, Week, Mini Maand, Maand en Jaar weergave
8	Dagen Kiezen
9	Evenementen delen (airdrop, mail, sms, whatsapp, Due, appigo Todo, Things, OmniFocus)

Tabel 3. Minimale Globale functionaliteit WeekCal framework

In tabel 3 staat beschreven welke functionaliteit minimaal in het framework moet worden gebouwd, ieder punt bestaat net zoals in de voorgaande tabel ook weer uit meerdere kleine delen functionaliteit; deze werk ik verder uit in de requirements. Zoals de titel van tabel 3 aanduidt is dit de globale functionaliteit en is dus niet gelijk aan de exacte requirements van het WeekCal framework, de requirements(Bijlage) heb ik echter wel uit deze tabel gehaald. De afgevallen functionaliteit staat in tabel 3 onder het kopje afgevallen functionaliteit, deze functionaliteit is afgevallen omdat het

afhankelijk was van een bepaalde target. Om het generieke doel van het WeekCal framework te behouden moest deze functionaliteit afvallen.

Na het opstellen van de requirements heb ik deze besproken met mijn afstudeer mentor en vervolgens geprioriteerd. Er zijn meerdere methodes om requirements te prioriteren en om de
juiste te kiezen heb ik naar de ervaringen van eerdere projecten van WeekCal gevraagd. Hieruit is de volgende tabel gekomen:

Methode	Gemiddeld aantal aanpassingen in prioriteit op 40 requirements
MOSCOW	6
Cijfers 0-10	14
Overig	14

Tabel 4. Fouten in prioriteit per prioriteer methode

In tabel 4 is duidelijk te zien dat MOSCOW de meest accurate methode is bij de projecten van WeekCal, om deze reden heb ik ook voor het WeekCal framework de MOSCOW methode gebruikt voor het prioriteren van de requirements.

De structuur waarop ik de requirements vervolgens heb opgesteld is:
Identificatie nummer - Beschrijving - Functioneel/Niet functioneel - Prioriteit.
In figuur 6 staat een voorbeeld afgebeeld.

Identificatie nummer	Beschrijving	Functioneel of niet functioneel	Prioriteit
R1	Het framework moet datums kunnen opmaken.	F	M

Afbeelding 6. Structuur Requirement

Tot slot heb ik alle belangrijke requirements verwerkt in use-case diagrammen. Hierbij heb ik gekeken naar de manier waarop de functionaliteit in de huidige Week Calendar app werkt en dit omgezet naar use-cases. Ik hiervoor gekozen om beter bekend te worden met de huidige situatie en de manier waarop functionaliteit van begin tot einde werkt [28]. Alle use-case diagrammen zijn terug te vinden in het requirements rapport.

4.1.2 Keuze projectmanagement -en systeemontwikkeling methode

Projectmanagement methode
De projectmanagement methode is de manier waarop het project wordt gepland en uitgevoerd. Het is gebruikelijk om binnen een bedrijf te werken met de standaard projectmanagement methode omdat bepaalde documenten dan kunnen worden hergebruikt en kennis al aanwezig is.

Bij WeekCal was er geen standaard projectmanagement methode maar wel een bepaalde werkwijze die van alle methodes het meest op SCRUM lijkt, omdat er Agile (mogelijkheid tot verandering binnen project) wordt gewerkt. Dit is een van de redenen dat ik als projectmanagement methode heb gekozen voor SCRUM, een striktere methode zoals Prince2 (waarbij een project niet Agile is) zou namelijk voor problemen kunnen zorgen. Naast het feit dat SCRUM het dichtste bij WeekCal ligt heeft SCRUM ook nog andere voordelen die goed met het WeekCal framework project passen, namelijk:

- Iedereen doet alles: Bij SCRUM doet ieder projectlid alle taken (analyseren, ontwerpen, bouwen, testen, documenteren). Tijdens het WeekCal project ben ik het

enige projectlid van het team en moet ik iedere taak uitvoeren, dit past dus goed bij SCRUM.
- Sprint Review: Bij SCRUM wordt na iedere sprint (periode van 2-3 weken waarin een stuk werkende code wordt ontwikkeld) een review gehouden. Bij het ontwikkelen van een framework is het gemakkelijk om te werken met losse testbare modules die kunnen worden afgerond tijdens sprints. Deze modules zijn dan dus losse stukken werkende code die goed kunnen worden gereviewed.
- Al ervaring met SCRUM, ik heb al meerdere jaren ervaring met SCRUM door school projecten met SCRUM uit te voeren.

Tot slot is SCRUM een van de makkelijkste projectmanagement methodes die het minste overhead geeft. Aangezien ik mijn opdracht alleen ging uitvoeren was een projectmanagement methode die veel werk kost (documentatie) niet meer het verbeterde management waard. Omdat scrum weinig extra tijd kost kon ik het gebruiken, maar eigenlijk is een projectmanagement methode bij een project met één lid overbodig.

Systeemontwikkeling methode

Na te hebben gekozen voor SCRUM als projectmanagement methode moest ik kiezen voor een systeemontwikkeling methode. Ik heb een kort onderzoek gedaan naar beschikbare methodes en vervolgens de meest passende methode gekozen, namelijk: Test-Driven Development (TDD). De redenen dat ik voor deze methode heb gekozen zijn:
- Niet bekend met codebase
  Dit betekent dat code snel fout kan worden geïnterpreteerd wat leidt tot fouten. Doordat bij TDD eerst de test wordt geschreven en vervolgens de code zullen deze fouten worden opgevangen. (door middel van een gefaalde unit-test)
- Hoge kwaliteit code [23]
  Het WeekCal framework zal wanneer het in productie gaat direct miljoenen gebruikers hebben. Dit betekent dat fouten in het project snel naar voren zullen komen en problemen kunnen veroorzaken. Hoge kwaliteit code door unit-testing voorkomt veel van deze problemen.
- Past goed bij SCRUM
  Iedereen doet alles, eerst unit tests maken en daarna code schrijven past er dus perfect bij.
- Diepe integratie van unit-testing in Xcode [24]
  Xcode heeft een geïntegreerd unit-testing framework genaamd XCTest. XCTest maakt het gemakkelijk om unit-tests bij te houden en kan zelfs code coverage bijhouden.

Een andere systeemontwikkeling methode die ik heb overwogen om te gebruiken is RUP. Omdat na de initiatie fase het duidelijk was hoe het eindproduct er uit zou gaan zien was een Agile systeemontwikkeling methode overbodig. RUP is een iteratieve systeemontwikkeling methode die goed past bij het gefaseerd werken dat ik eerder in dit hoofdstuk heb beschreven en is niet Agile.

Uiteindelijk heb ik toch gekozen voor Test Driven Development omdat het principe van eerst testen en vervolgens bouwen voor het project belangrijker was dat wel of niet Agile werken. Wel heb ik het principe van gefaseerd werken uit RUP geleend en gerealiseerd als de framework ontwikkeling en implementatie fases.

4.2 Framework Ontwikkeling

Na de initiatie fase van het WeekCal framework project kon ik beginnen aan de ontwikkeling fase. Het doel van deze fase is het realiseren van de eerder opgestelde requirements naar een functioneel, getest en gedocumenteerd WeekCal framework.

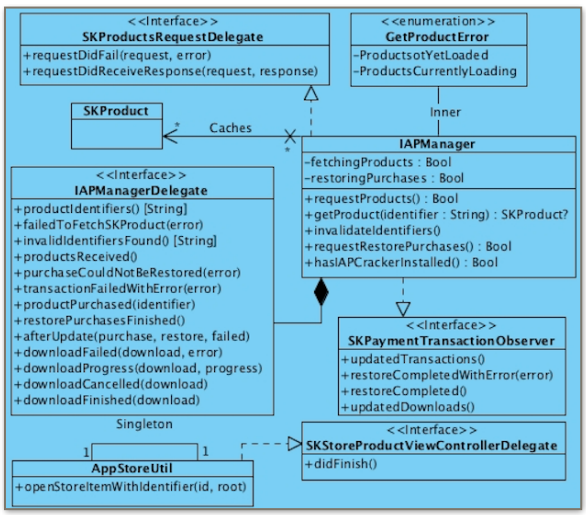
Zoals beschreven in de opdrachtomschrijving bestond het project in eerste instantie uit het omzetten van Objective-C code uit de Week Calendar app naar Swift in een nieuw framework genaamd 'WeekCal framework'. Na de code en het ontwerp van de Week Calendar app te bestuderen ben ik tot de conclusie gekomen dat een volledig nieuw ontwerp maken een beter idee was dan het oude ontwerp over te nemen. Het oude ontwerp was namelijk ontworpen met Objective-C limieten, niet generiek en niet gedocumenteerd. Met het vernieuwde ontwerp moest ook vernieuwde code worden geschreven om het ontwerp te realiseren, dit is waar ik dus niet oude code heb omgezet maar volledig nieuwe code heb geschreven.

Om dit doel te bereiken heb ik eerst een system design diagram ontworpen, vervolgens het design in functionele en unit-tested code omgezet en tot slot het framework gedocumenteerd. Deze activiteiten worden in dit hoofdstuk verder uitgewerkt.

4.2.1 System Design Diagram

Zoals eerder beschreven in hoofdstuk 4.1.1 is het van groot belang dat het WeekCal framework generiek wordt ontworpen. Om ervoor te zorgen dat dit doel bereikt kan worden heb ik een generiek system design diagram ontworpen als basis van het WeekCal framework, dit is een overzicht van classes, onderlinge relaties en verantwoordelijkheden. Met gebruik van een system design diagram is het mogelijk om slecht bedachte ontwerpen snel op te merken, problemen in het WeekCal framework kunnen op die manier worden voorkomen nog voordat er code wordt geschreven. Zonder een system design diagram zullen deze problemen minder snel opvallen en moet code vaak worden herschreven, wat tijd en moeite kost. Hiernaast wordt code vaak lelijk wanneer deze voor een specifiek probleem moet worden herschreven, er wordt dan namelijk vaak niet nagedacht of de oplossing generiek is.

Het system design diagram voor het WeekCal framework heb ik ontworpen met de tool Visual Paradigm community 11.0, dit is een gratis tool die gebruikt kan worden om UML diagrammen te ontwerpen. Binnen het system design diagram wordt er onderscheid gemaakt tussen verschillende zelfstandige modules, deze worden aangeduid met 'packages' die binnen het diagram eruit zien als grote blauwe vierkanten. In afbeelding 7 is de StoreKit module afgebeeld, deze module is afgeleid uit functionaliteit 4 van tabel 2; de module verzorgt dus in-app aankopen. Ik heb als doel gesteld dat iedere module volledig onafhankelijk en generiek moet zijn, dit zorgt voor:



Afbeelding 7. Voorbeeld Module (StoreKit)

- Hoge cohesie
- Lage Koppeling
- Makkelijker om framework modulair te implementeren.

Zoals te zien is in afbeelding 7 is de module volledig zelfstandig en is het ontwerp niet target specifiek (en dus generiek). Naast het generieke ontwerp in afbeelding 7 is er nog een ander principe te zien, namelijk: Design Patterns. Later in dit verslag wordt per module het ontwerp toegelicht en gemotiveerd.

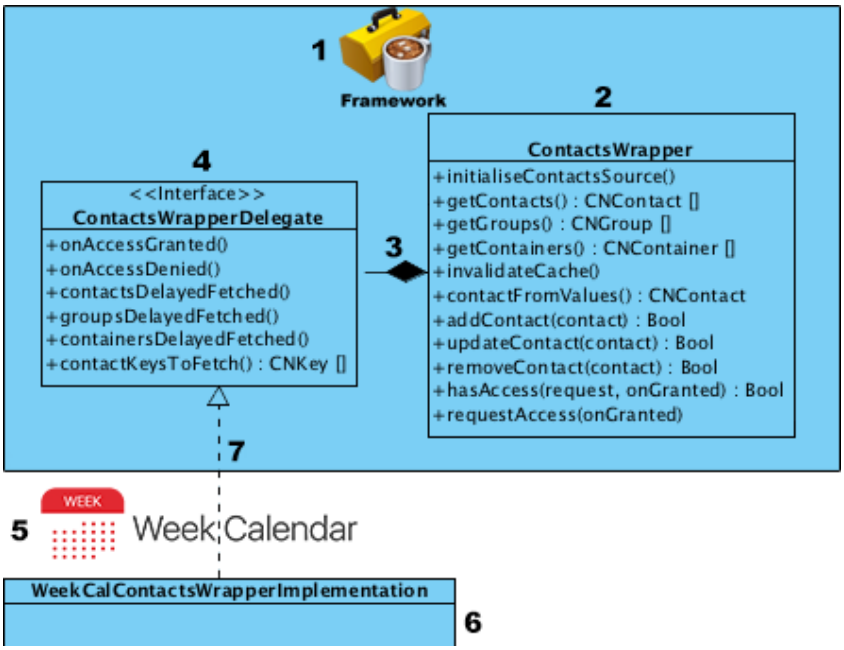
4.2.1.1 Ontwerp Technieken

4.2.1.1.1 Design Patterns

Een design pattern is een generiek opgezette softwarestructuur die een bepaald veelvoorkomend software-ontwerpprobleem oplost. Een dergelijk design pattern biedt geen concrete oplossing voor ontwerp problemen, maar biedt een template dat gebruikt kan worden om een ontwerp probleem op te lossen. Binnen het WeekCal framework wordt er gebruik gemaakt van een aantal design patterns die er samen voor zorgen dat het WeekCal framework generiek, net, uitbreidbaar en gebruiksvriendelijk blijft. De gebruikte design patterns worden hieronder verder beschreven.

Strategy

Binnen het WeekCal framework wordt er veel gebruik gemaakt van het strategy design pattern. Het strategy design pattern is een methode die gebruikt kan worden om een object in runtime specifiek gedrag te geven. Om een framework echt generiek te maken moet een bepaald target (diegene die het framework implementeert) het gedrag van het framework kunnen veranderen.



Afbeelding 8. Strategy Pattern in WeekCal framework

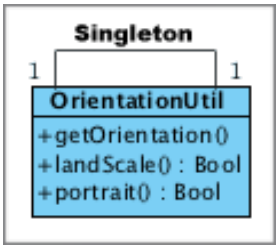
In afbeelding 8 is het strategy pattern schematisch afgebeeld, dit is slechts een van de vele implementaties van het strategy pattern in het WeekCal framework.
Zoals te zien is bestaat de implementatie in afbeelding 8 uit 2 delen, namelijk het framework (1) en het target (diegene die het framework implementeert) (5). Binnen het framework zijn in een standaard implementatie twee classes, namelijk de klasse waarvan het target(5) het gedrag wilt kunnen aanpassen (2) en een interface dat het aanpasbare gedrag van (2) definieert (4). Het target (5) kan vervolgens een klasse aanmaken die het gewenste gedrag uitvoert (6), deze klasse moet een implementatie zijn van het interface waarin het aanpasbare gedrag staat gedefinieerd (4). Tot slot kan het target (5) vertellen aan de klasse waarvan het gedrag moet worden veranderd (2) dat het de gedragsklasse van het target (6) moet gebruiken.

De klasse met het aanpasbare gedrag(2) weet exact wat het kan verwachten van de gedragsklasse(6) omdat deze een implementatie is van het interface waarin het gedrag staat gedefinieerd (4). Op deze manier kan het framework verschillend gedrag uitvoeren bij verschillende targets zonder dat het zelf veranderd hoeft te worden, dit

draagt sterk bij aan hoe generiek het framework is. Als het target specifieke gedrag namelijk in het framework opgenomen had moeten worden zou dit ten koste zijn gegaan van het generieke ontwerp.

Het strategy pattern kwam in de oude codebase ook vaak voor, maar exclusief als implementatie (6). De iOS SDK forceert namelijk bij user interfaces en meerdere van haar frameworks het gebruik van het strategy pattern. In geen enkel geval was in de oude codebase het strategy pattern volledig gebruikt om een eigen klasse (en dus niet uit de iOS SDK) aangepast gedrag te geven, met het WeekCal framework wordt hier verandering in gebracht.

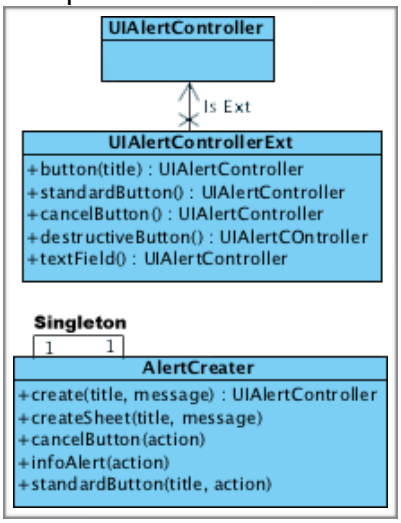
Singleton
Singleton is een design pattern dat het aantal objecten van een bepaalde klasse tot één beperkt. Op deze manier is het mogelijk om er zeker van te zijn dat er slechts één object is van een bepaalde klasse en dat alle toegang tot de betreffende klasse via hetzelfde object verloopt. Het singleton pattern wordt in het WeekCal framework vaak gebruikt bij klassen die specifieke functionaliteit hebben die hetzelfde blijft bij ieder object.



Afbeelding 9. Singleton in WeekCal Framework

In afbeelding 9 staat afgebeeld op welke wijze het singleton pattern wordt gebruikt in het WeekCal framework. De klasse 'OrientationUtil' biedt 3 methodes die niet afhankelijk zijn van variabelen, dit betekend dat het gedrag van de methodes hetzelfde is bij ieder object. In dit geval kan er dus gebruik gemaakt worden van een singleton object voor toegang tot de klasse OrientationUtil.

Builder & Chaining (fluent interface)
Het builder design pattern is een pattern dat wordt gebruikt om nieuwe objecten van klassen aan te maken. Het doel van het builder pattern is het verminderen van parameters in klasse constructors en hiermee verantwoordelijkheid van functionaliteit verspreiden tussen methodes.



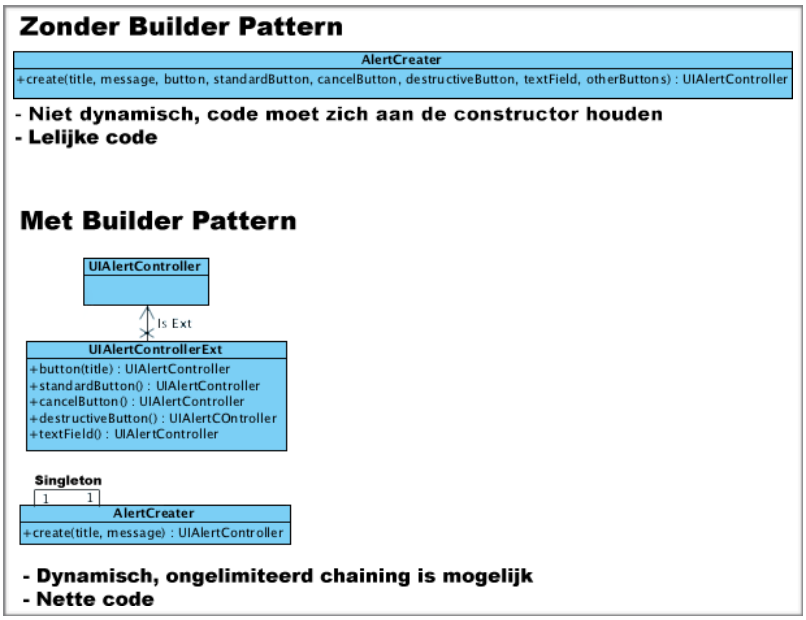
Afbeelding 10. Builder pattern in WeekCal Framework

In afbeelding 10 zijn twee design patterns te herkennen, namelijk het singleton en het builder pattern. Naast deze design patterns wordt er ook gebruik gemaakt van extensions (*hoofdstuk 4.2.1.1.2*) om het builder pattern mogelijk te maken.

Het builder pattern begint in afbeelding 10 bij de klasse 'AlertCreater', deze klasse heeft de functie create. De create methode maakt een nieuwe UIAlertController aan, de UIAlertController klasse beschikt echter niet over een builder pattern implementatie, bevind zich in een afgeschermd framework en kan niet worden aangepast. Om deze reden wordt er gebruik gemaakt van een extensie (UIAlertControllerExt) om de builder functionaliteit toe te voegen aan de UIAlertController klasse.

Binnen de UIAlertController extension staat onder anderen de functie button(title), deze functie voegt eerst een knop toe aan de alert en retourneert vervolgens weer het UIAlertController object. Op deze manier kan vervolgens de geretourneerde UIAlertController weer gebruikt worden om bijvoorbeeld de methode cancelButton of destructiveButton aan te roepen. Dit principe, dat ook wel methode chaining of fleunt interface wordt genoemd, maakt code korter en leesbaarder. In dit geval wordt het builder pattern dus gerealiseerd met gebruik van een fluent interface.

In afbeelding 11 staat het verschil tussen een design met het builder pattern tegenover een design zonder het builder pattern afgebeeld.



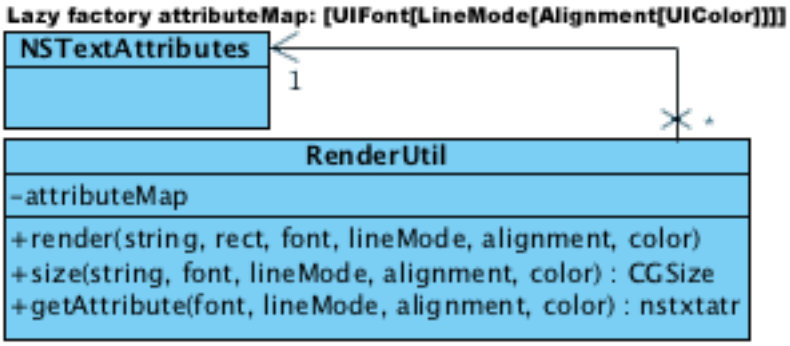
Afbeelding 11. Builder VS niet builder

Zoals in afbeelding 11 te zien is moeten er bij het builder pattern wel meer klassen worden gemaakt dan zonder het builder pattern, meer klassen betekend echter niet dat de code minder leesbaar wordt. In het bovenstaande voorbeeld is het met het builder pattern mogelijk om ongelimiteerd knoppen toe te voegen (button, cancelButton, etc), waar het design zonder builder pattern gelimiteerd is aan de constructors van de klasse. Tot slot kan in het voorbeeld de code om knoppen toe te voegen worden hergebruikt op een later moment, constructors kunnen daarentegen alleen gebruikt worden om nieuwe objecten te maken en kunnen dus niet vanuit iedere situatie worden gebruikt. De oude codebase maakte nooit gebruik van het builder pattern of fluent interfaces en stond dan ook vol met ontwerpen die sterk leken op de bovenste uit afbeelding 11.

Lazy Initialization
Lazy initialization is een design pattern met als doel dat objecten pas worden aangemaakt wanneer deze nodig zijn en niet eerder. In het WeekCal framework wordt dit pattern meerdere malen gebruikt maar dit is niet terug te vinden in het system design diagram. De reden hiervoor is omdat de programmeertalen Swift en Objective-C beiden support hebben voor lazy variabelen waardoor er geen extra design voor ontworpen hoeft te worden. De oude codebase gebruikte ook lazy initialization, een aanzienlijk gedeelte van de lazy initialization code kon ik dus omzetten naar Swift. Hiernaast heb ik ook waar het pattern goed uitkomt het toegepast.

Lazy Factory
Lazy factory is een design pattern met als doel het centraliseren en cachen van object creatie van een bepaalde klasse. Het verschil tussen het reguliere Factory design pattern en lazy factory is dat bij lazy factory objecten met bepaalde variabelen worden opgeslagen zodat deze weer kunnen worden opgehaald en niet opnieuw hoeven te worden gecreëerd.

In afbeelding 12 staat de klasse RenderUtil afgebeeld, deze klasse is verantwoordelijk voor het tekenen van strings op het scherm van iPhones en iPads. Om een string te tekenen moet er eerst een 'string attributes' object worden aangemaakt, bij 60FPS zou dit per string 60 keer per seconde moeten gebeuren. Het is dan dus duidelijk dat er gebruik gemaakt kan worden van het lazy factory pattern voor het aanmaken en ophalen van string attributes. In afbeelding 12 wordt dit afgebeeld als de relatie naar NSTextAttributes met een vierdimensionale map; één dimensie per parameter zodat zonder door heel de map heen te zoeken een NSTextAttribute kan worden opgehaald.



Afbeelding 12. Lazy factory

Al deze design patterns zorgen samen voor een net, uitbreidbaar en generiek ontwerp voor het WeekCal framework. Dit is een grote verbetering ten opzichte van het oude ontwerp van de Week Calendar app dat geen gebruik maakt van de Builder en Lazy Factory design patterns en het Strategy pattern niet optimaal gebruikt.

4.2.1.1.2 Overige Ontwerp technieken en keuzes

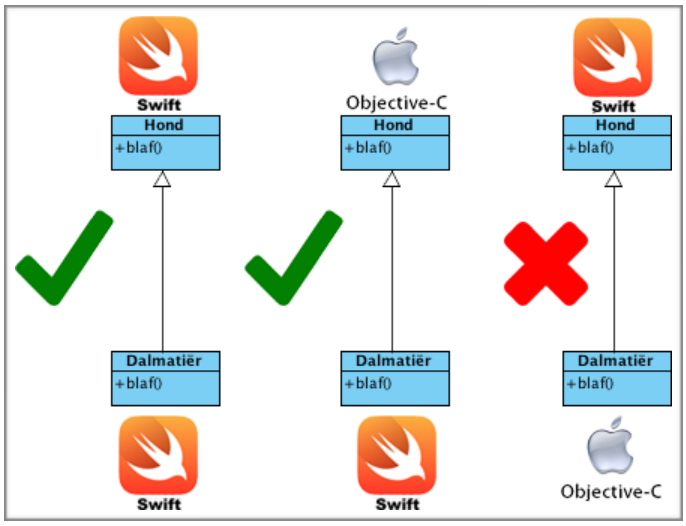
Naast het gebruiken van design patterns heb ik ook andere technieken en keuzes gemaakt die niet onder een specifiek design pattern vallen. Deze keuzes en technieken worden hieronder toegelicht en gemotiveerd.

Waarom composition (interface implementaties) en geen inheritance?

Er zijn een aantal redenen waarom ik heb gekozen om te werken met interfaces en implementaties:

- Een klasse kan van meerdere interfaces implementeren maar slechts één superklasse hebben. [6]
- Werkt met het veel toegepaste strategy pattern
- Het is niet mogelijk om in Objective-C subklassen te maken van klassen die in Swift zijn geschreven. Dit betekent dat als bijvoorbeeld in het WeekCal framework de klasse Hond wordt aangemaakt is het niet mogelijk om in de Week Calendar app een klasse Dalmatiër te maken als subklasse van de klasse Hond. Dit komt doordat het WeekCal framework in Swift is geschreven en de Week Calendar app in Objective-C. Dit is de primaire reden dat inheritance zo veel mogelijk wordt vermeden in het WeekCal framework. In afbeelding 13 staat afgebeeld wat er wel en niet mogelijk is qua inheritance.

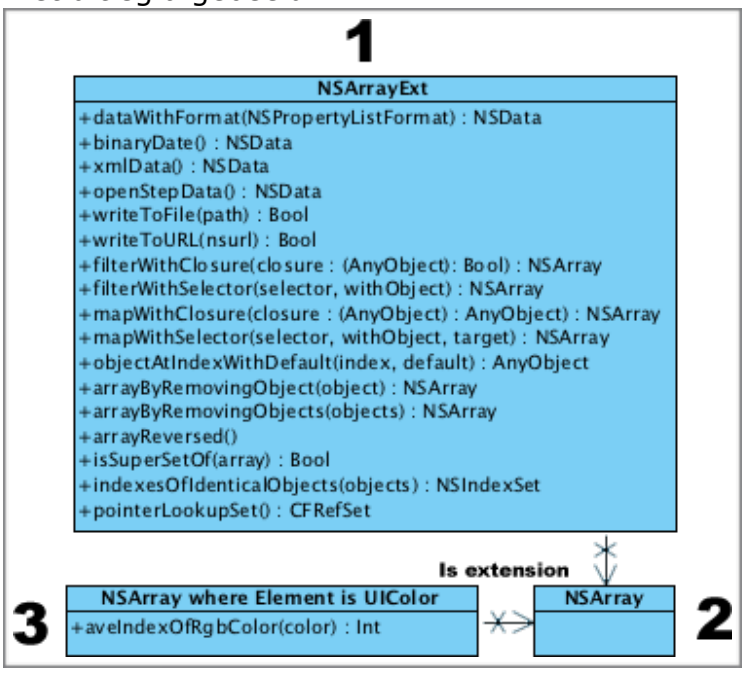
  Er is wel een andere mogelijkheid die de afwezigheid van inheritance gedeeltelijk kan vullen, namelijk: Extensions.



Afbeelding 13. Limieten inheritance Swift & objective-c

Extensions

Extensions zijn zoals de naam al doet denken uitbreidingen. In zowel Swift als Objective-C is het mogelijk om een zogenaamde extensie klassen te maken, een dergelijke extension kan de functionaliteit van een andere klasse uitbreiden. De manier waarop extensions functioneren wordt later in dit verslag uitgewerkt maar om het system design diagram goed te kunnen begrijpen staat hieronder een voorbeeld met uitleg afgebeeld.

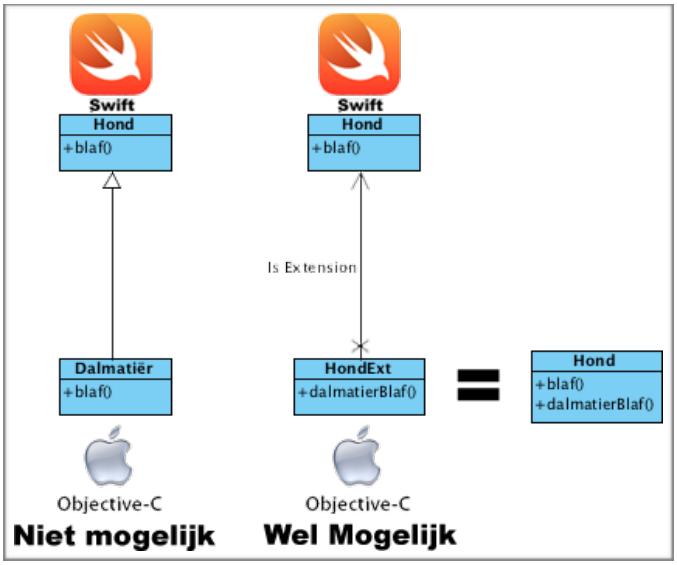


Afbeelding 14. Extension binnen WeekCal Framework

In afbeelding 14 staan 3 klassen afgebeeld, 1 normale klasse (2) en twee extension klassen (1, 3). Zowel '*NSArrayExt*' als '*NSArray where…*' breiden klasse (2) uit, het is dus mogelijk om meerder extensie klassen te hebben voor één normale klasse. Op deze manier is het mogelijk om het 'god class' probleem in enige zin te voorkomen, de uitgebreide klasse heeft alsnog veel functionaliteit maar wel verspreid in extensie klassen. Het verschil tussen '*NSArrayExt*' en '*NSArray where…'* is dat '*NSArrayExt*' altijd 'NSArray' uitbreidt waar '*NSArray where…*' dit alleen doet als het betreffende NSArray object bestaat uit UIColor objecten. UIColor kan worden vervangen met ieder willekeurig object type wanneer dit nodig is, heel handig dus om verantwoordelijkheden vast te leggen.

Binnen afbeelding 14 staan ook twee relatie lijnen (UML), deze lijnen zijn alleen ter verduidelijking gebruikt om aan te geven dat het gaat om extensie klassen en welke klasse er wordt extended. De lijnen betekenen dus niet dat bijvoorbeeld *NSArrayExt* een object van *NSArray* bijhoudt.

In afbeelding 15 staat globaal beschreven op welke wijze extensies een vervanging zijn voor inheritance. Op de linkerzijde van de afbeelding staat de klasse *Dalmatier* als subklasse van *Hond. Wanneer* Hond in Swift is aangemaakt en Dalmatier in Objective-C wordt het ontwerp niet meer ondersteunt door iOS, dit is een limiet van de bridging functionaliteit van iOS.



Afbeelding 15. Waarom extensies een oplossing bieden voor de afwezigheid van inheritance

Het is wel mogelijk om in Objective-C een Swift klasse te extenden, dit staat afgebeeld in de rechterzijde van afbeelding 15. De functie *dalmatierBlaf* is met dit ontwerp beschikbaar in de klasse hond vanuit zowel Swift als Objective-C context. Hoewel dit geen echte vervanging voor inheritance is, is het wel een oplossing waar extra functionaliteit nodig is in plaats van specifieker gedrag. Interfaces blijven zoals eerder beschreven de beste oplossing, extensions zijn slechts een handige aanvulling.

Waarom Singleton en niet Static

Zoals eerder beschreven geeft het singleton pattern toegang tot een bepaalde klasse via exact één object, niet meer en niet minder. Een statische methode is een methode die aan een klasse toebehoort en niet een object. Statische methodes kunnen dus worden uitgevoerd zonder dat er eerst een object van de klasse moet worden aangemaakt. Zowel singleton als statische methodes geven dus een gemakkelijke toegang tot bepaalde functies/methoden en worden in eerste instantie voor hetzelfde doeleinde gebruikt, het singleton pattern heeft echter een aantal belangrijke voordelen:

- Singleton objecten worden pas aangemaakt wanneer deze voor de eerste keer worden aangeroepen, statische functies worden altijd geladen. [7.4]
- Statische functies leiden tot meer statische functies en statische variabelen, dit leidt tot niet generieke code en is bad-practice.
- Het singleton pattern gebruikt een klasse met methodes en variabelen en kan dus [7.2]:
  - worden uitgebreid met de eerder beschreven extensions.
  - Interfaces implementeren [7.1]

  - Inheritance gebruiken [7.3]
  - Een singleton object kan worden meegegeven als parameter [7.2]
- Statische methodes kunnen deze functies niet gebruiken

Naamgeving
Het diagram bevat veel expliciete namen, zoals bijvoorbeeld '*transactionFailedWithError*' in de StoreKit module. Er zijn drie soorten namen in het ontwerp:
- Namen die ik zelf heb gekozen
  - Duidelijke namen die lang mogen zijn. [8]
- Namen die zijn afgeleid uit de documentatie die door Apple is geleverd voor alle Apple frameworks [5]
  - Namen die gebruikt moeten worden voor iOS interface implementaties
- Namen die zijn afgeleid uit de codebase van de Week Calendar app
  - Beginnen vaak met ave, dit zijn specifieke functies en zijn opgenomen in het design om de verantwoordelijkheden van klassen duidelijk te houden.

4.2.2 Master testplan

Een belangrijk aspect van het WeekCal framework project is de testbaarheid ervan. Het zomaar testen van een applicatie werkt vaak bij kleine applicaties prima, maar bij het WeekCal framework, waar enorm veel(circa 1 miljoen/maand) gebruikers afhankelijk van zullen zijn, is dit niet genoeg. Inconsistenties binnen tests, zwak geteste onderdelen die sterk getest moeten worden en vice versa zullen altijd opstaan zonder een goed plan, ofwel, een master testplan.

Om deze reden heb ik voor het WeekCal framework een master testplan opgesteld volgens TMap [4]. Ik heb voor de methode van TMap gekozen omdat deze bij veel IT'ers bekend is en omdat ik er zelf al ervaring mee had. Binnen een master testplan wordt vastgelegd wat er getest moet worden, wie wat test en wanneer, hoe er getest wordt en hoeveel tijd er besteed kan worden aan testen. Aan de hand van deze informatie kan er met vertrouwen een planning voor het testen van het WeekCal framework worden opgesteld. Deze planning loopt in het geval van het WeekCal framework project synchroon met de globale planning omdat er met scrum en TDD wordt gewerkt. Dit betekent dat voordat er code wordt geschreven er eerst een test wordt gemaakt waardoor de planning voor de ontwikkeling van het WeekCal framework gelijk loopt aan de test planning. Het volledige master testplan is in de bijlage te vinden.

4.2.3 WeekCal Framework

Na het vastleggen van alle afspraken, waaronder wat er gemaakt gaat worden, hoe dit bereikt gaat worden en op welke wijze dit getest wordt kon ik beginnen aan de ontwikkeling van het WeekCal framework.

Om te starten met het WeekCal framework project heeft WeekCal mij de huidige Week Calendar iPhone app, geschreven in Objective-C, aangeleverd. De codebase van de Week Calendar app was mij tot dit moment volledig onbekend wat niet handig was voor een aantal requirements, het volgende dat ik dus moest doen was het bekend worden met de codebase. De manier waarop ik dit heb gedaan is door telkens een requirement te bekijken en vervolgens in de codebase deze functionaliteit op te zoeken. Ik heb hierbij zo min mogelijk hulp gevraagd van mijn afstudeer mentor of andere iOS ontwikkelaars binnen WeekCal zodat ik zelfstandig bekend kon worden met de codebase. Pas wanneer ik ergens niet uit kwam, of gewoon opheldering over een bepaalde functie nodig had heb ik om hulp gevraagd.

Taal	Classes	Regels Code
Objective-C	640	123.728
C/C++ Header	655	12.021
C++	2	5.346
Swift	20	655
Objective-C++	2	421
Totaal	1.319	142.171

Tabel 5. Analyse Week Calendar app

In tabel 5 staat schematisch beschreven hoeveel code er in de uitgangssituatie bestond in de WeekCal framework applicatie. Zoals te zien is in tabel 5 bestaat de volledige Week Calendar app uit 142.171 regels aan code, dit is natuurlijk teveel om allemaal om te zetten naar Swift code. Om deze reden worden de requirements uitgewerkt door volledig nieuwe Swift code te schrijven die voldoet aan de requirements, vervolgens worden specifieke functies uit de Week Calendar app omgezet van Objective-C naar Swift om tot slot in het WeekCal framework te verwerken. In hoofdstuk 4.3.2 wordt beschreven hoeveel regels code er nu minder zijn voor dezelfde functionaliteit.

4.2.3.1 Aanpak framework ontwikkeling

Zoals eerder beschreven in dit verslag heb ik gekozen om te werken met SCRUM en test-driven development, beide methodes zijn natuurlijk slechts een collectie van ideeën en methodieken om projecten beter te laten verlopen. Tijdens het WeekCal framework project heb ik zo veel mogelijk de standaard protocollen gevolgd in hoeverre dit voordelig voor het project was. De volgende aspecten van SCRUM heb ik volgens de standaard methode uitgevoerd:

- Product backlog
  - Aan de start van de ontwikkeling heb ik een volledige product backlog gemaakt met de requirements uit het requirements rapport. De requirements zijn opgenomen als user stories.

- Sprints
  - Tijdens het ontwikkelen van het WeekCal framework heb ik consistent in sprints gewerkt waarin ik telkens een bepaald aantal requirements heb gerealiseerd.
- Sprint backlog
  - Voor iedere sprint heb ik een sprint backlog aangemaakt volgens de standaard SCRUM methode
- Iedereen doet alles
  - Dit principe heb ik bij nadruk uitgevoerd. Ik heb alles zelf ontworpen, geprogrammeerd en getest van de sprint backlog per sprint.

Hiernaast heb ik de volgende aspecten op een andere (of niet) manier uitgevoerd:
- Sprint review
  - Tijdens sprint reviews had ik alleen tijdens de implementatie fase een visueel product. De sprint reviews bestonden daardoor uit een code review, korte test rapportage en adviezen over wat beter kan / wat goed gaat. De sprint review heb ik altijd uitgevoerd met mijn afstudeer mentor Saurabh Garg.
- Daily standup
  - Ik was samen met Saurabh Garg (mijn afstudeer mentor) het enige lid van het WeekCal framework project. Saurabh Garg was er niet elke dag waardoor een daily standup niet nuttig was. Wel was er een 'weekly standup' in WeekCal waarin iedereen over zijn week en de volgende vertelt. Deze standup hielp bij het motiveren om door te werken, omdat ik anders weinig te vertellen had tijdens de standup.

De systeemontwikkeling methode test-driven development heb ik volledig volgens protocol gevolgd. Voor iedere functie heb ik eerst een unit test geschreven en vervolgens de betreffende code, hierover wordt later in het verslag meer verteld. Zoals eerder beschreven heb ik met SCRUM sprints gewerkt, in iedere sprint heb ik één, een deel of meerdere modules ontwikkeld. Iedere module bestaat uit een bepaalde hoofdfunctionaliteit, zoals bijvoorbeeld contacten ophalen. Binnen dit hoofdstuk zal iedere module verder worden uitgewerkt.

4.2.3.2 Opzet Project

Om te beginnen met de ontwikkeling van het WeekCal framework moest ik het eerst aanmaken met XCode (hoofdstuk 2.4.2). In plaats van een standaard project aan te maken moest ik kiezen tussen een static framework of dynamic Swift framework. Omdat static frameworks geen Swift ondersteunen was de keuze van een Swift dynamic framework (hoofdstuk 4.2.3.4.1) snel gemaakt.

4.2.3.2.1 Swift versie

Swift is een snel groeiende taal warvan versie 2.3 momenteel in productie is. In laat September 2016 wordt Swift 3 uitgerold en kunnen applicaties er in worden omgezet om de nieuwe functies en veranderingen te kunnen gebruiken.

Hoewel Swift 3 nog niet in productie is zijn er wel al vroege beta versies van Swift 3 verkrijgbaar wat het mogelijk maakt om het WeekCal framework direct in Swift 3 te ontwikkelen. Samen met mijn stagementor heb ik gediscussieerd over de keuze tussen 2.3 en 3, hieruit is gekomen dat het WeekCal framework voor nu in Swift 2.3 geschreven moet worden omdat:
- Swift 3 is in beta en zal veel veranderen, kost extra werk
- Conversie van Swift 2.3 naar 3 zal niet veel werk kosten (automatische migratie is mogelijk) [26]
- Code gemaakt met Beta software van Apple kan niet worden uitgegeven [27]

4.2.3.3 Modules

In dit hoofdstuk worden de modules van het WeekCal framework beschreven. Per module worden:

- De requirements afgebeeld
- Het ontwerp toegelicht
- Lastige onderdelen bij de implementatie toegelicht
- Gemaakte keuzes gemotiveerd en problemen toegelicht

4.2.3.3.1 Contacts Module

De contacts module verzorgt alle communicatie tussen het WeekCal framework en de contacten van de gebruiker.

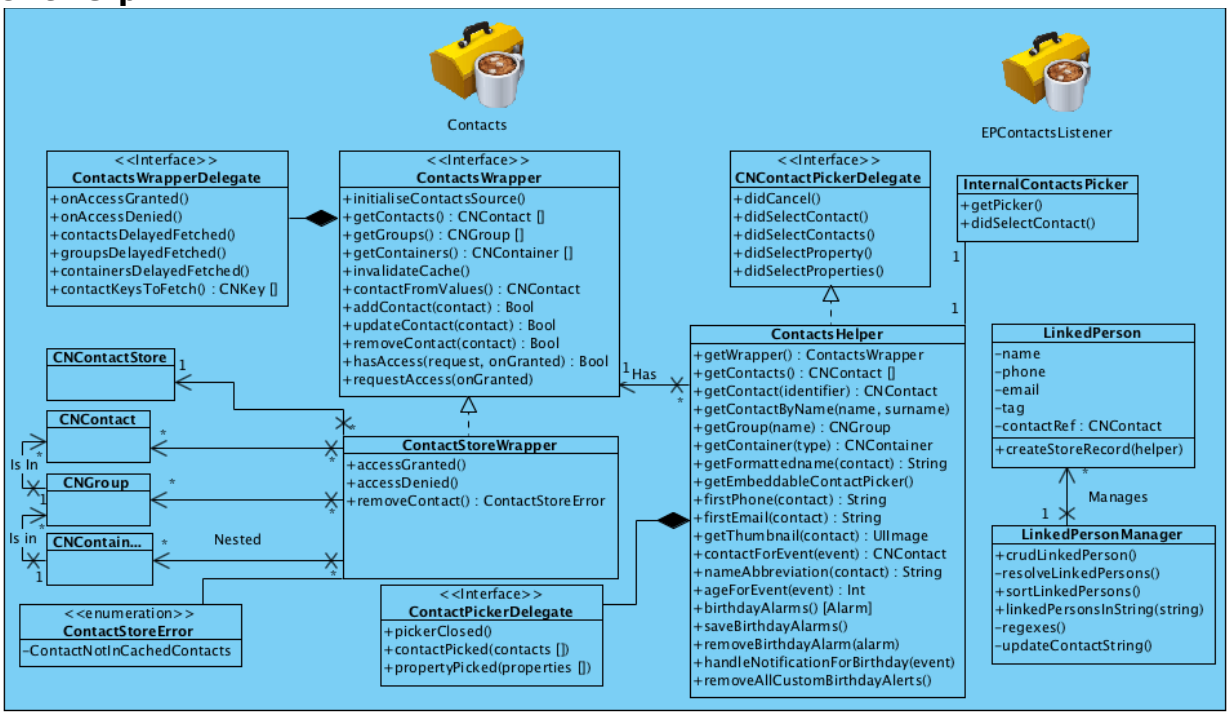
Requirements

De volgende requirements vallen onder de Contacts module:

Requirements Nummer	Beschrijving
R37	Het framework moet contacten, groepen en containers uit de contacts app kunnen ophalen
R38	Het framework moet thumbnail images kunnen maken voor contacten zoals iOS 9 dat doet in de contacts app
R39	Het framework moet contacten kunnen zoeken op naam, email, telefoonnummer of closure
R40	Het framework moet de Contact Picker kunnen opstarten en erop reageren
R41	Het framework moet de contact controller kunnen opstarten en erop reageren
R42	Het framework moet de verjaardagen van contacten kunnen detecteren; en deze omzetten in EKEvents en notificaties
R46	Het framework moet contacten binnen 1 seconde kunnen ophalen

Tabel 6. Contacts module requirements

Ontwerp



Afbeelding 16. Ontwerp Contacts Module

De hoofdklassen in de contacts module zijn *ContactsWrapper* en *ContactsHelper*. *ContactsWrapper* definieert de functionaliteit die nodig is voor communicatie met Apple's EventKit, van *ContactsWrapper* wordt op dit moment één implementatie ondersteunt genaamd *ContactsStoreWrapper*. *ContactsHelper* is verantwoordelijk voor overige functies die met contacten te maken hebben, hiernaast implementeert ContactsHelper ook *CNContactPickerDelegate* waarmee contacten uit een lijst worden gekozen door de gebruiker. Door twee klassen te gebruiken in plaats van één wordt de verantwoordelijkheden gescheiden en het god klasse probleem verminderd.

In het ontwerp van de contacts module komt één design pattern twee keer voor, namelijk het strategy pattern. *ContactsWrapperDelegate* definieert het aanpasbare gedrag van *ContactsWrapper* en *ContactPickerDelegate* het gedrag van *ContactsHelper, gedrag* implementaties komen in dit ontwerp niet omdat het gedrag pas bij de framework implementatie fase ontwikkeld wordt. Een groot voordeel van het strategy pattern in dit ontwerp is dat het tegelijk gebruikt kan worden als notificatie systeem voor asynchrone processen (user input).In *ContactsWrapperDelegate* staat bijvoorbeeld de functie *onAccessGranted*, deze functie kan vervolgens worden geïmplementeerd en als het ware op de hoogte worden gesteld van wat er binnen het framework gebeurd.

Toelichting Module

Voordat de Contacts module zijn primaire functies kan uitvoeren moet er eerst toegang tot de contacten van de gebruiker worden verkregen. Om toegang te verkrijgen moet er gebruik gemaakt worden van het Contacts framework [9.1] dat door Apple is aangeleverd.

Contacts en AddressBook

Het contacts framework is een nieuw framework en wordt niet gebruikt in de huidige Week Calendar app, in plaats van het Contacts framework gebruikt de Week Calendar app het AddressBook framework [10]; de oude variant van Contacts. Om deze reden was het bij de Contacts module onmogelijk om code uit de Week Calendar app te hergebruiken, de verschillen tussen Contacts en AddressBook zijn namelijk groot.

```
func contactWithName(name: String) -> CNContact
{
    let predicate = CNContact.predicateForContactsMatchingName(name)
    let keysToFetch = [CNContactFormatter.descriptorForRequiredKeysForStyle(.FullName), CNContactPhoneNumbersKey]
    let contact = CNContactStore().unifiedContactsMatchingPredicate(predicate, keysToFetch: keysToFetch).first
    return contact
}
```

Afbeelding 17. Contact met naam ophalen met Contacts framework

In afbeelding 17 wordt een contact met zijn/haar volledige naam opgehaald aan de hand van zijn/haar naam. Eerst wordt een predikaat aangemaakt die duidelijk maakt welke contacten er moeten worden opgehaald (*let predicate…*), vervolgens wordt er gedefinieerd welke informatie er per contact nodig is (*let keysToFetch…*) en tot slot wordt aan het Contacts framework gevraagd om het contact op te halen aan de hand van het predikaat en de benodigde informatie (*let contact…*). Het contact is hierna volledig toegankelijk voor het WeekCal framework.

Bij het AddressBook framework is dezelfde functionaliteit behalen een stuk lastiger:

```
func contactWithName(name: String) -> ABRecordRef?
{
    let addressBookRef: ABAddressBook = ABAddressBookCreateWithOptions(nil, nil).takeRetainedValue()
    let allContacts = ABAddressBookCopyArrayOfAllPeople(addressBookRef).takeRetainedValue() as Array
    for record in allContacts {
        let contactName = ABRecordCopyCompositeName(currentContact).takeRetainedValue() as String
        if(contactName == name) {
            return contact
        }
    }
    return nil
}
```

Afbeelding 18. Contact met naam ophalen in AddressBook framework

In afbeelding 18 wordt exact hetzelfde gedaan als in afbeelding 17, echter is het direct al duidelijk dat er meer code voor nodig is. Een aantal grote verschillen zijn:

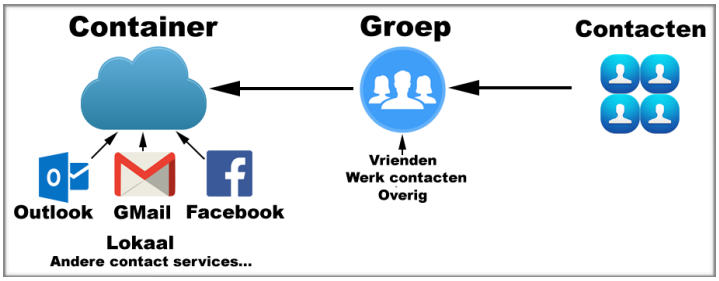
- AddressBook werkt met zwakke pointers, dit betekend dat wanneer er iets wordt gevraagd aan het AddressBook framework het altijd een pointer naar een object zal retourneren. Een pointer kan vervolgens worden omgezet naar een object door *takeRetainedValue* uit te voeren. Hier zijn wel een aantal nadelen aan, *takeRetainedValue* kan ieder willekeurig type object retourneren, het weet namelijk niet wat voor object er vast zit aan zijn pointer. Om deze reden moet er bij *takeRetainedValue* altijd worden gecast, dit is te zien in afbeelding 18 (as Array, as String).

  Hiernaast zijn pointers niet thread safe, als door een ander thread het geheugen waar de pointer naartoe wijst verandert en vervolgens het het object van de pointer wordt opgevraagd zal de applicatie crashen zonder duidelijke reden (moeilijk te debuggen). Het Contacts framework werkt altijd met sterke types en zonder pointers.
- AddressBook heeft geen ondersteuning voor predikaten, je moet dus zelf de contacten filteren.

Het AddressBook framework is gemaakt voor Objective-C en het Contacts framework voor Swift, dit brengt voor het WeekCal framework één lastig probleem. De Week Calendar app die uiteindelijk het WeekCal framework zal gaan gebruiken verwacht input van het AddressBook framework, dat volledig met pointers werkt. Het Contacts framework zal echter nooit pointers retourneren waardoor de Week Calendar app in eerste instantie foute input zal krijgen. Dit probleem kan worden opgelost door alle contact code in de Week Calendar app compatible te maken met het WeekCal framework wat verder wordt uitgewerkt in hoofdstuk 4.3.1.1.

Structuur Contacten [9.3]

Binnen iOS bestaat de structuur van contacten uit drie delen; Containers, groepen en contacten. Een container is waar contacten in eerste instantie vandaan komen, dit kan de telefoon zelf zijn (lokaal) maar ook een externe bron zoals Outlook of GMail. Vervolgens worden de contacten binnen een container verdeeld in groepen. Deze groepen kan de gebruiker zelf aanmaken, maar kunnen ook door externe bronnen of derde partijen worden aangemaakt voor de gebruiker.



Afbeelding 19. Structuur contacten in iOS

Het Contacts framework geeft volledige toegang tot al deze componenten, WeekCal gebruikt echter alleen contacten en niet groepen of containers. Helaas moet het framework alle containers en groepen eerst ophalen om bij de contacten te komen (tenzij er een predikaat wordt gebruikt). Zoals in afbeelding 19 staat afgebeeld begint de toegang bij container, vervolgens bij groep en tot slot bij contacten.

Om contacten op te kunnen halen is er echter nog één ding nodig, namelijk: toegang [9.2]. Het WeekCal framework kan alleen bij contacten komen als de gebruiker expliciet op zijn toestel hier toegang voor geeft, dit werkt door een popup te laten zien waar er naar toegang wordt gevraagd. Om toegang vragen en de input van de gebruiker brengen de volgende moeilijkheid met zich mee:

- Zodra er wordt gevraagd om input wordt de code asynchroon. Er moet namelijk gewacht worden op input van de gebruiker, er moet dus code worden geschreven dat op ieder moment juist kan reageren op de gegeven input. De oplossing die ik hiervoor heb gebruikt is de *closures techniek*. De keuze voor deze techniek wordt geforceerd door het Contacts framework omdat dit framework ook gebruik maakt van *closures*.

Contact Picker

Naast toegang verkrijgen was er om de contacts module compleet te maken nog één probleem waar ik een oplossing voor moest vinden. Het Contacts framework biedt een zogenoemde 'Contacts Picker', dit is een user interface waarin de gebruiker een contact kan uitkiezen om hier later acties mee uit te voeren. De contacts picker uit het Contacts framework werkt op zichzelf perfect maar heeft één gebrek waardoor het voor aantal cases onbruikbaar werd in het WeekCal framework: Het is niet mogelijk om de Contacts Picker te embedden, dit betekend dat de contacts picker alleen als zelfstandig user interface kon werken en niet binnen een ander interface.

Afbeelding 20. Embedded en zelfstandig

In afbeelding 20 staan zowel een zelfstandig als een embedded contact picker afgebeeld, het zelfstandige interface werkt binnen de Week Calendar app als er alleen een contact moet worden gekozen. Er zijn echter ook gevallen waar dit niet werkt, in

afbeelding 20 staat zo’n dergelijke case waar er snel gekozen moet kunnen worden tussen de contact picker, recente en een aangepaste kleur. In dat geval moest de contact picker embedded worden wat de contact picker uit het Contacts framework niet toelaat. Om dit probleem te overkomen moest ik of zelf een contacts picker bouwen of een contact picker van een derde partij gebruiken, om tijd te besparen heb ik gekozen om eerst te zoeken naar een contact picker van een derde partij; hiervoor heb ik de volgende eisen opgesteld.

- Moet contacten kunnen uitzoeken (multi-selection en single selection)
- Moet gebruik maken van het Contacts framework (en dus niet AddressBook)
- Moet look en feel hebben van de contact picker uit het Contacts framework
- Moet embedding ondersteuning
- MIT Licentie

Hier kwamen twee kandidaten uit: EVContactsPicker [1] en EPContactsPicker [2]. Beiden bieden op kleine verschillen na dezelfde functionaliteit maar EPContactsPicker lijkt veruit het meest op de contacts picker uit het Contacts framework. Om deze reden heb ik gekozen om EPContactsPicker te integreren in het WeekCal framework.

Om software van derde partijen te integreren in iOS applicaties (of frameworks) is er een speciaal systeem door Apple ontwikkeld, genaamd: CocoaPods (hoofdstuk 4.2.3.3.3). Een andere mogelijkheid was het direct toevoegen van de software van de derde partij door het te slepen in het WeekCal framework, bij CocoaPods is het echter makkelijker om software te updaten, toevoegen of verwijderen.

Een nadeel van CocoaPods is dat het projecten complexer maakt qua configuratie, wat voor slechts één stuk software (in dit geval) teveel overhead kan geven. Software van derde partijen is echter vaak weer afhankelijk van andere software, een probleem dat CocoaPods automatisch oplost [11]. Hiernaast is het makkelijk om in de toekomst nog meer software toe te voegen via CocoaPods. Om deze redenen heb ik ervoor gekozen om CocoaPods te gebruiken.

4.2.3.3.2 StoreKit Module

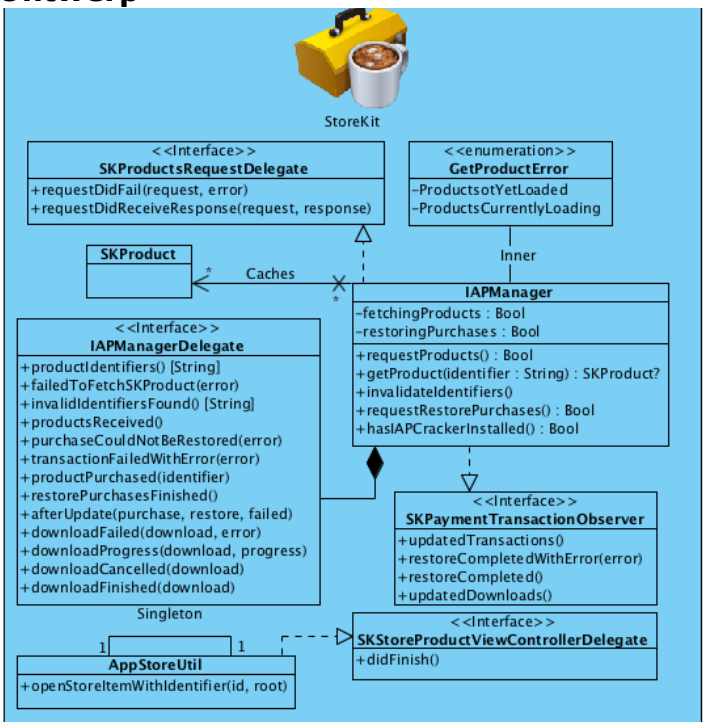
De StoreKit module is verantwoordelijk voor het opvragen, kopen en herstellen van in-app aankopen. In-app aankopen zijn producten die binnen de app kunnen worden gekocht en zitten dus niet in de aankoop van de Week Calendar app zelf. Om deze reden worden in-app aankopen dus niet volledig gemanaged door de app stores, maar moeten ze door de app zelf worden gestart.

Requirements
De volgende requirements vallen onder de StoreKit module:

Requirements Nummer	Beschrijving
R43	Het framework moet in-app purchases kunnen opvragen van iTunes
R44	Het framework moet in-app purchase aankopen kunnen starten
R45	Het framework moet aankopen herstellen kunnen starten

Tabel 7. Requirements StoreKit

Ontwerp



Afbeelding 21. Ontwerp StoreKit Module

De StoreKit module is qua ontwerp niet zeer uitgebreid en bestaat uit twee hoofdklassen, IAPManager en IAPManagerDelegate. IAPManager verzorgt alle communicatie met het StoreKit framework van Apple en implementeert daarom ook '*SKProductRequestDelegate*' en '*SKPaymentTransactionObserver*'. IAPManager houdt ook alle '*SKProducts*' bij zodat de verantwoordelijkheid over communicatie tussen het StoreKit framework en het WeekCal framework volledig en zelfstandig door IAPManager wordt gemanaged.

Om de StoreKit module generiek te houden wordt er gebruik gemaakt van het strategy pattern, '*IAPManager*' kan via het interface '*IAPManagerDelegate*' informatie over transacties doorgeven aan de implementatie, hiernaast kan de implementatie ook aangeven welke producten er geladen moeten worden ('*productIdentifiers*'). Het

strategy pattern wordt dus net zoals in de Contacts module gebruikt voor gedrag en als notificatie systeem.

Toelichting Module
De StoreKit is de enige module waar er met aankopen wordt omgegaan en is daardoor ook de module met het hoogste risico, er wordt namelijk met de aankopen en dus het geld van de gebruiker gewerkt. Om het risico binnen de module te verlagen moest de code duidelijk, overzichtelijk en net worden geschreven zodat er geen misverstanden konden ontstaan. Apple zelf biedt ook een framework aan dat, net zoals deze module, StoreKit heet. Het StoreKit framework is een uitgebreid getest en veilig framework en is de enige manier om de gebruiker producten te laten kopen. Het is dan ook logisch dat de StoreKit module het StoreKit framework gebruikt om de requirements te realiseren. In feite is de StoreKit module een tussenlaag tussen de Week Calendar app en het StoreKit framework, de module verzorgt bijvoorbeeld fouten management, houdt producten bij en maakt de communicatie met het StoreKit framework makkelijk.

De StoreKit module heeft 3 mogelijke toestanden, namelijk idle, fetchingProducts en ready. Zodra de StoreKit module wordt geïnitialiseerd veranderd de toestand van idle naar fetchingProducts. De StoreKit module vraagt nu de te verkopen producten op van Apple, zodra de producten binnen zijn worden deze opgeslagen en gaat de toestand over naar ready. In de ready toestand kan de code uit afbeelding 22 worden uitgevoerd, oftewel, er kunnen in-app purchases worden gedaan. In afbeelding 22 wordt eerst een product opgevraagd, de functie zal alleen een product terugsturen als alle producten zijn geladen en het opgevraagde product bestaat.

```
public func getProduct(identifier : String) throws -> SKProduct?
{
    if(fetchingProducts) {
        throw GetProductError.ProductsCurrentlyLoading
    }
    guard let _products = products else {
        throw GetProductError.ProductsNotYetLoaded
    }
    for(product) in _products {
        if(product.productIdentifier == identifier) {
            return product
        }
    }
    return nil
}
public func requestPurchaseProduct(product : SKProduct) -> Bool
{
    if(SKPaymentQueue.canMakePayments()) {
        let payment = SKPayment(product: product)
        SKPaymentQueue.defaultQueue().addPayment(payment)
        return true
    } else {
        return false
    }
}
```

Afbeelding 22. Product opvragen en kopen

```
if let product = try getProduct("com.week_calendar.subscribe") {
    if(requestPurchaseProduct(product)) {
        //Gebruiker kan nu inloggen en vervolgens de aankoop doen
    }
}
```

Afbeelding 23. In-app purchase starten

Om een aankoop te starten hoeft de gebruiker van het framework nu alleen het te kopen product op te vragen via de "*getProduct*" functie en kan vervolgens "*requestPurchaseProduct*" uitvoeren met het opgevraagde product. Als de gebruiker toestemming heeft om in-app purchases uit te voeren wordt de aankoop gestart, de aankoop wordt verder door Apple geregeld totdat de aankoop is voltooid en het framework hiervan op de hoogte wordt gesteld door Apple.

Om het framework op te hoogte te kunnen stellen heeft het StoreKit framework een observer nodig. Een observer is een klasse die een bepaald interface implementeert, hierdoor weet het StoreKit framework welke functionaliteit het kan uitvoeren van de observer. Zodra een aankoop procedure is voltooid wordt de volgende functie uitgevoerd in de observer: *"paymentQueue(queue, transactions)"*, in afbeelding 24 staat deze functie afgebeeld zoals hij in het WeekCal framework wordt geïmplementeerd.

```
public func paymentQueue(queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction])
{
    for(transaction) in transactions
    {
        let state = transaction.transactionState
        switch(state)
        {
        case SKPaymentTransactionState.Purchased:
            delegate.productPurchased?(transaction, identifier: transaction.payment.productIdentifier)
            queue.finishTransaction(transaction)
            break
        }
        /* ... */
    }
}
```

Afbeelding 24. Aankopen observeren

In afbeelding 24 worden alle veranderde transacties aan het WeekCal framework doorgegeven, om de aankoop flow die eerder in dit deze module is gestart af te maken wordt er gekeken of de staat van iedere transactie "*Purchased*" is. Wanneer dit het geval is wordt de aankoop doorgegeven aan het target (diegene die het framework implementeert) doormiddel van het strategy pattern en wordt de transactie voltooid.

4.2.3.3.3 EventStore Module

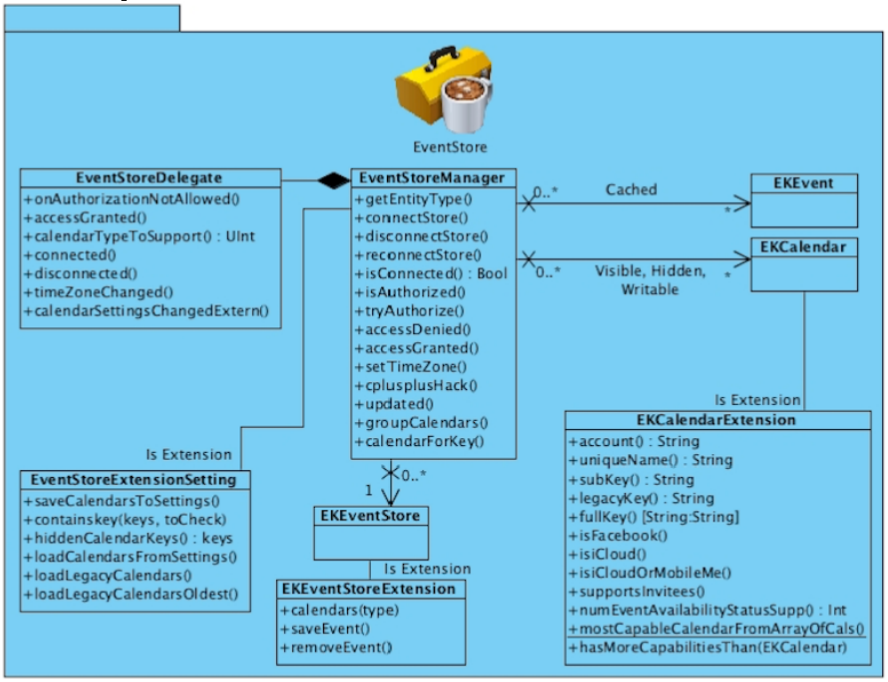
De EventStore module is verantwoordelijk voor alle communicatie tussen het WeekCal framework en de kalenders van de gebruiker, deze module heeft veel overeenkomsten met de contacts module en heeft een vergelijkbare structuur. De reden hiervoor wordt in dit hoofdstuk verder uitgewerkt.

Requirements
De volgende requirements vallen onder de EventStore module:

Requirements Nummer	Beschrijving
R17	Het framework moet calendars kunnen ophalen
R18	Het framework moet calendars zichtbaar en onzichtbaar kunnen maken
R19	Het framework moet EKEvents kunnen toevoegen/wijzigen/verwijderen aan calendars
R20	Het framework moet kunnen detecteren welke opties een calendar heeft, en of deze meer heeft dan een andere calendar.

Tabel 8. EventStore module requirements

Ontwerp



Afbeelding 25. Ontwerp EventStore Module

Net zoals bij de voorgaande modules wordt ook in de EventStore module het concept van een zelfstandige manager klasse in combinatie met het strategy pattern aangehouden. De klasse '*EventStoreManager*' managet in dit geval alle communicatie met het EventKit framework van Apple en houdt '*EKEvent*' en '*EKCalendar's*' bij (EventKit klassen). Het strategy pattern wordt in de EventStore module gerealiseerd met het interface '*EventStoreDelegate*' dat het gedrag van '*EventStoreManager*' beïnvloed. Net zoals in voorgaande modules wordt de gedragsklasse ook als notificatie systeem gebruikt.

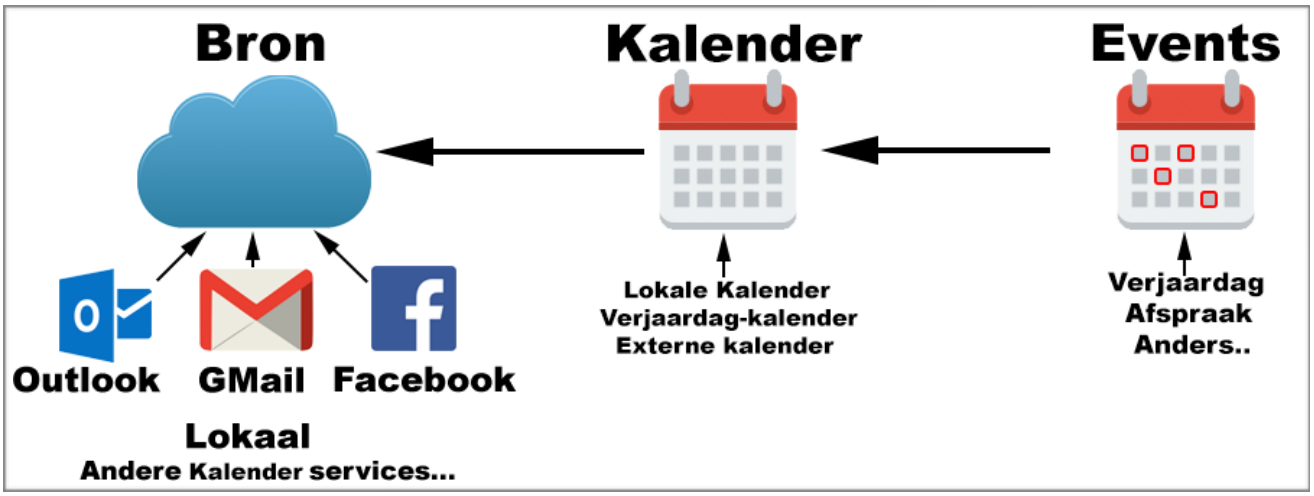
Naast het strategy pattern wordt er in deze module veel gebruik gemaakt van Extensions, binnen deze extensions wordt de functionaliteit van de oude Week

Calendar app geplaatst. Persoonlijk had ik liever helper klassen gemaakt om consistent te blijven met bijvoorbeeld de Contacts module, echter werd in het ontwerp van de Week Calendar app veel gebruikt gemaakt van extensies voor kalender functies. Al deze extensies om zetten in helper klassen had veel tijd gekost bij het bouwen en had slechts voor betere consistentie gezorgd, om deze reden heb ik het ontwerp met extensies aangehouden.

Toelichting Module
De Week Calendar applicatie is een agenda applicatie en maakt dus extensief gebruik van kalenders. Dit betekent dus ook dat de kalender module zeer uitgebreid is; hoewel de requirements er niet direct op duiden zit er veel functionaliteit binnen de EventStore module. Dit komt doordat er veel kalender functionaliteit binnen de Week Calendar app bestond die geporteerd moest worden naar het WeekCal framework, hieronder valt onder anderen het groeperen van kalenders, geschikte kalenders berekenen voor bepaalde events en nog een aantal andere functies waarvan de meest belangrijke later in dit hoofdstuk worden beschreven.

Om te communiceren met de kalenders van de gebruiker moet er gebruik worden gemaakt van het EventKit framework dat door Apple wordt geleverd. Dit werkt dus hetzelfde als bij de Contacts module waar er ook gecommuniceerd moest worden met een framework van Apple, hiernaast zijn er nog andere overeenkomsten tussen de Contacts en EventStore module:
- Allebei de modules moeten eerst toestemming van de gebruiker verkrijgen en gebruiken hier closures voor.
- Beide modules maken gebruik van een Apple framework en maken een abstractie laag tussen de Apple frameworks en de gebruiker van het framework.

Er is echter ook een groot verschil, in de Week Calendar app wordt op dit moment ook het EventKit framework gebruikt waar bij de contacts module er een verouderd framework werd gebruikt. Dit betekent dat veel functies van de Week Calendar app konden worden overgezet naar het WeekCal framework door de code om te zetten van Objective-C naar Swift (hoofdstuk 4.2.3.3.9); wel moest hier alsnog veel code van worden gemoderniseerd. Voordat ik code ben gaan omzetten heb ik eerst de requirements afgewerkt en geïmplementeerd zoals ik dit had ontworpen in het system design diagram.



Afbeelding 26. Structuur kalenders in iOS

Zoals in afbeelding 26 wordt afgebeeld komen alle kalenders van een gebruiker eerst uit een bron, deze bron kan of een kalender service zijn (zoals outlook) of lokaal. Het verschil tussen deze twee types is dat lokale kalenders alleen veranderd kunnen

worden op de gebruiker zijn telefoon, waar externe kalenders kunnen veranderen vanuit andere locaties [12]. Dit betekent dat de synchronisatie van externe kalenders lastiger is dan bij lokale kalenders, hiervoor is echter een goede oplossing die door Apple wordt aangeboden, namelijk: NSNotificationCenter (hoofdstuk 4.2.3.3.4)

Het ophalen, opslaan en verwijderen van kalenders is met gebruik van het EventKit framework zeer makkelijk. Het enige dat door het WeekCal framework gedaan moest worden was het vragen van toestemming en error management. Toestemming vragen voor kalenders werkt hetzelfde als bij de Contacts module; vervolgens kan met slechts enkele regels code de kalenders van de gebruiker worden opgehaald.

```
func calendarSummary()
{
    //Communicatie starten met EventKit
    let store = EKEventStore()
    //Kalenders ophalen
    let calendars = store.calendarsForEntityType(.Event)
    //Event Aanmaken
    let event = EKEvent(eventStore: store)
    event.calendar = calendars[0]
    event.title = "Test Event"
    //Event aanmaken of updaten
    do {
        try store.saveEvent(event, span: .ThisEvent)
    } catch {
        //Error Handling, Kon event niet opslaan
    }
    //Event verwijderen
    do {
        try store.removeEvent(event, span: .ThisEvent)
    } catch {
        //Error Handling, kon event niet verwijderen
    }
}
```

Afbeelding 27. Kalenders ophalen en event muteren

In afbeelding 27 wordt eerst een connectie gemaakt met Apple's EventKit. Vervolgens worden alle kalenders met het type Event opgehaald zodat de events ervan kunnen worden veranderd en uitgelezen. Hierna wordt een nieuw event aangemaakt (*Test Event)* en wordt er geprobeerd de event toe te voegen aan de gebruiker zijn kalender. (*store.saveEvent*). Tot slot wordt het event weer uit de kalender gehaald (*store.removeEvent)* en is de kalender weer in een originele staat. Een complexe functionaliteit als kalenders is met EventKit dus heel gemakkelijk en heeft slechts een paar regels code nodig.

Het makkelijkste onderdeel van de EventStore module waren de initiële requirements en het implementeren van het system design diagram, de lastige onderdelen waren echter de functies die geporteerd moesten worden van de Week Calendar app naar het WeekCal framework. De belangrijkste van deze functies zijn gemodelleerd in het ontwerp van de EventStore module en bevinden zich exclusief in de extensie klassen. Een voorbeeld van een omgezette functie is te vinden in hoofdstuk 4.2.3.4.9.

4.2.3.3.4 Theming Module

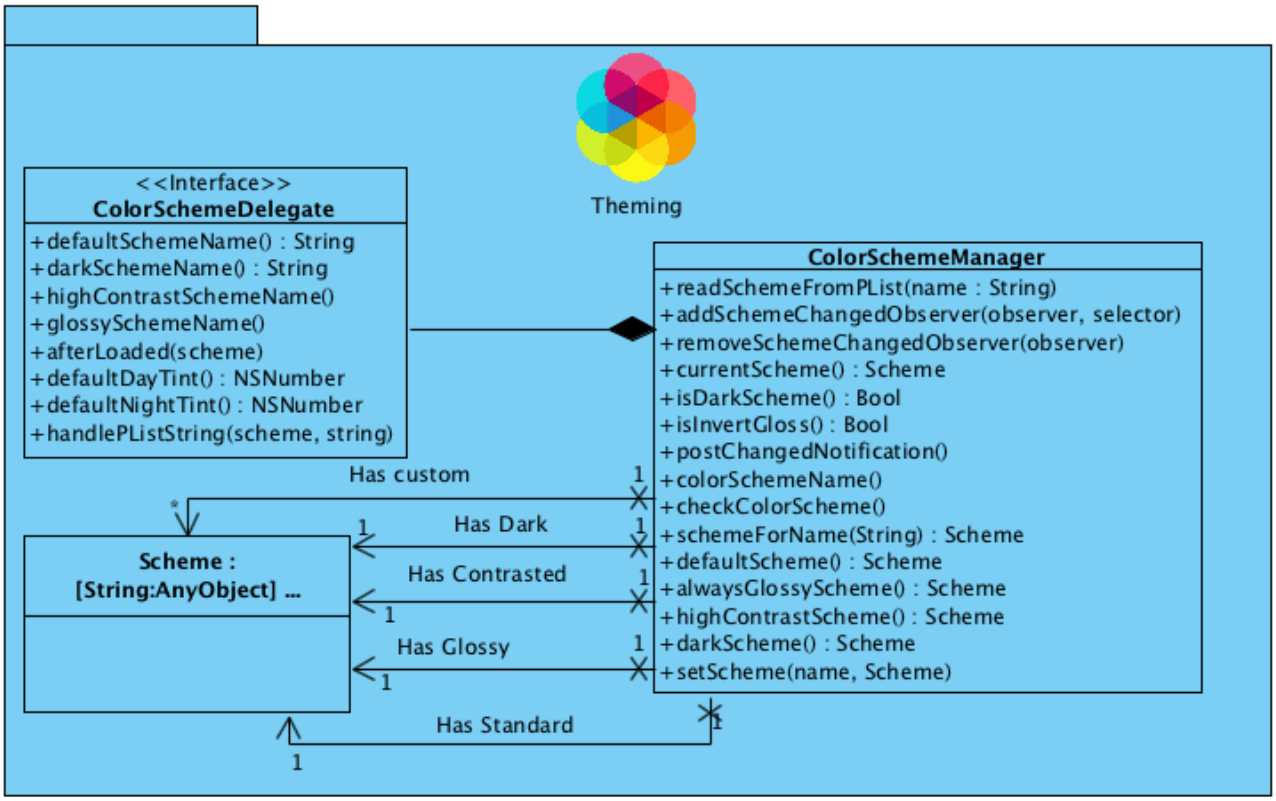
De theming module is verantwoordelijk voor standaardisatie van kleurenschema's en alle berekeningen met kleuren. Deze module is een van de minst generieke omdat er met specifieke kleurenschema's wordt gewerkt, toch heb ik ervoor gekozen om deze module op te nemen in het WeekCal framework. In dit hoofdstuk wordt verder beschreven waarom ik hiervoor heb gekozen.

Requirements
De volgende requirements vallen onder de Theming module:

Requirements Nummer	Beschrijving
R14	Het framework moet kleuren schema's kunnen lezen uit property lists
R15	Het framework moet met kleuren kunnen rekenen. (r,g,b aanpassen, luminance, saturatie, grayscale)
R16	Het framework moet kunnen detecteren of een kleur licht of donker is

Tabel 9. Requirements Theming Module

Ontwerp

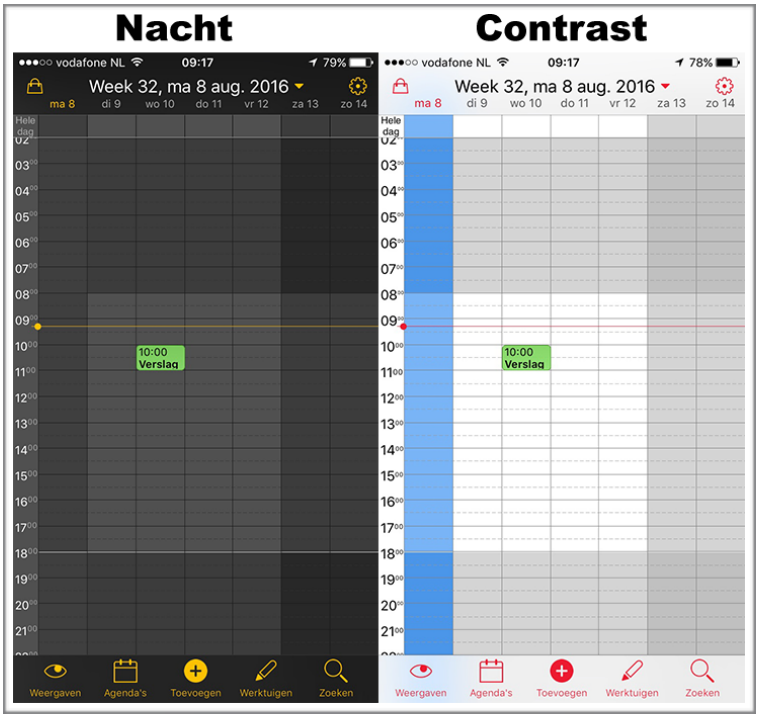


Afbeelding 28. Ontwerp Theming Module

Het ontwerp van de Theming module is voor een deel nog hetzelfde als het ontwerp van de Week Calendar app. Het grootste verschil is dat ik via het strategy pattern de module generiek heb gemaakt, '*ColorSchemeDelegate*' kan met het nieuwe ontwerp worden gebruikt om '*ColorSchemeManager*' specifieke kleuren schema's te laten laden. In het geval van deze module had het strategy pattern ook vervangen kunnen worden met variabelen in '*ColorSchemeManager*' maar om de module consistent te houden met eerdere modules heb ik ook hier het strategy pattern toegepast.

'*ColorSchememanager*' houdt '*Scheme*' objecten bij, Scheme is in dit geval geen eigen klasse maar een typealias (hoofdstuk 4.2.3.4.10) en wordt gebruikt om kleuren schema's mee bij te houden. In het oude ontwerp werd er geen typealias gebruikt en was het ook niet mogelijk om oneindig schema's bij te houden ('*has custom*').

Toelichting Module
Kleuren schema's zijn collecties van kleuren die door de Week Calendar applicatie gebruikt kunnen worden om een bepaalde look and feel te krijgen. Zo zijn er bijvoorbeeld het contrasted en nacht kleuren schema.



Afbeelding 29. Color schemes in de app

Omdat kleuren schema's bijna altijd app specifiek zullen zijn is het anti-generiek om deze schema's op te nemen in het WeekCal framework. Wat ik wel heb ontwikkeld in de theming module is de structuur om kleuren schema's in te kunnen laden en gebruiken. Om kleuren schema's in te laden wordt er in het WeekCal framework gebruikt gemaakt van property lists (plist), dit zijn bestanden met een XML structuur waarin normaal gesproken instellingen worden opgeslagen. In een plist kunnen alle basis data types worden opgeslagen (String, Bool, NSNumber) en ook types zoals NSData (voor binaire data zoals foto's) en NSDate (voor datums).

Het type UIColor, wat nodig is om kleuren te bepalen, wordt echter niet direct ondersteund door plists. Voor de kleuren schema's is het UIColor type het meest belangrijke type; om toch via plists UIColor's in te kunnen laden heb ik gebruik gemaakt van hex waardes (hoofdstuk 4.2.3.4.6)

Zoals eerder beschreven levert het WeekCal framework alleen de structuur om kleuren schema's in te laden, het framework vraagt de plists op van het target en laadt deze vervolgens in. Om het framework te testen en compatible te houden met de Week Calendar app heb ik alvast de kleuren schema plists gemaakt voor de thema's van de Week Calendar app. In afbeelding 30 staat een klein gedeelte van het kleuren schema voor het nacht thema. Er wordt telkens een sleutel gegeven met eraan vast een

Root	Dictionary	(69 items)
isDarkScheme	Boolean	YES
monthTodayShadowMarkersColor	String	#000000
monthTodayText	String	#ffffff
seperator	String	#000000
fullLine	String	#1a1a1a
monthOutsideShadowMarkersColor	String	#000000
monthDarkenShadowMarkersColor	String	#000000
monthWeekNumber	String	#b3b3b3
monthWeekNumberShadow	String	#000000
modern_monthTodayText	String	todayTextColorDark
modern_agendaTodayBackground	String	#3f3f3f

Afbeelding 30. Nacht kleuren schema

hex kleur code, ook is het mogelijk om in plaats van hex codes bepaalde woorden te gebruiken zoals '*todayTextColorDark*' om aan te geven dat de standaard dag kleur gebruikt moet worden. Dit scheelt veel werk wanneer het thema van een app veranderd moet worden en geeft de mogelijkheid om standaard kleuren dynamisch te veranderen binnen het target.

Kleuren schema's zijn een goede manier om een applicatie aanpasbaar te maken voor de gebruiker, het betekend echter ook dat kleuren niet altijd hetzelfde zullen zijn. Als er bijvoorbeeld een kleur zwart (of donkerbruin) wordt gemaakt en er zwarte tekst is op het nu donkere vlak is de tekst niet meer te lezen. Om deze reden heeft de Theming module ook functies waarmee kleuren kunnen worden verlicht en het contrast tussen kleuren berekend kan worden. Er kan hiermee worden gedetecteerd of een kleur leesbaar is boven een andere kleur, vervolgens kan het WeekCal framework dan een meer contrasterende en dus beter leesbare kleur kiezen voor de tekst. Het is daardoor mogelijk om iedere kleurencombinatie te kiezen zonder dat de app onbruikbaar wordt.

4.2.3.3.5 CoreLocation Module

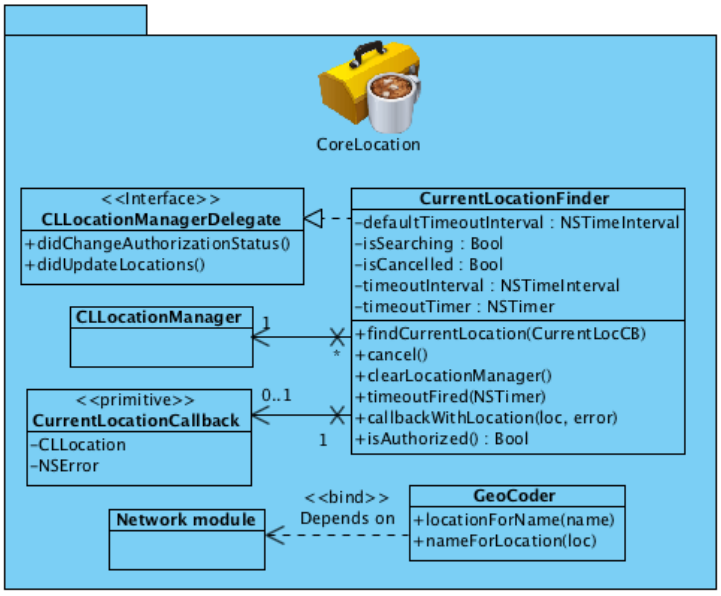
De CoreLocation module is verantwoordelijk voor het ophalen van de gebruiker zijn locatie, dit is qua requirements de kleinste module. Het ophalen van de gebruiker zijn locatie is echter niet zo gemakkelijk als het lijkt.

Requirements
De volgende requirements vallen onder de CoreLocation module:

Requirements Nummer	Beschrijving
R32	Het framework moet de huidige locatie van de gebruiker kunnen bepalen

Tabel 10. Requirements CoreLocation Module

Ontwerp



Afbeelding 31. Ontwerp CoreLocation Module

Het ontwerp van de CoreLocation module is simpel en consistent met voorgaande modules. '*CurrentLocationFinder*' verzorgt alle communicatie met Apple's frameworks. Het strategy pattern wordt toegepast binnen de module als implementatie, CurrentLocationFinder implementeerd namelijk CLLocationManagerDelegate voor locatie updates.

Net zoals in de Theming module wordt er ook in deze module gebruik gemaakt van een TypeAlias, namelijk: '*CurrentLocationCallback*', ditmaal heeft de typealias echter ook parameters. Normaal gesproken zou ik gebruik maken van het strategy pattern voor een dergelijke callback design, de oude week calendar codebase gebruikte echter voor deze functionaliteit een ontwerp dat herschreven had moeten worden als ik het strategy pattern had gebruikt. Om deze reden heb ik ervoor gekozen om de oude callback methode te gebruiken maar wel te verbeteren met gebruik van een typealias.

Het laatste dat opvalt in deze module is de GeoCoder klasse, het is namelijk de enige klasse die afhankelijk een andere module. De GeoCoder klasse heeft de Network module nodig om zijn functionaliteit uit te voeren en maakt hiermee het module onafhankelijkheid doel onmogelijk, een betere oplossing was wellicht het plaatsen van de GeoCoder klasse in de Network module. GeoCoding hoort echter meet bij de CoreLocation module dan bij de Network module en is daarom ook onderdeel van de CoreLocation module ten koste van de cohesie.

Toelichting Module
Om de locatie van de gebruiker te achterhalen moet er eerst toegang worden gevraagd aan de gebruiker. Het vragen van toegang werkt hetzelfde als bij de Contacts en EventStore modules, er wordt een toegang aanvraag uitgevoerd met een closure die voor verdere acties zorgt zodra er toegang is gegeven. Zodra er toegang is gegeven door de gebruiker kan Apple's LocationManager worden gestart en updates geven, deze updates worden via het Strategy pattern bekend gemaakt aan de CoreLocation module.

```
public func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation])
{
    callback?(locations[0])
}
```

Afbeelding 32. Locatie updates

Zodra in Apples LocationManager een nieuwe locatie huidige locatie binnenkomt wordt in het WeekCal framework de functie uit afbeelding 32 uitgevoerd. De functie callback is niet een functie binnen de CoreLocation module maar een variabele. Het target kan deze variabele toewijzen en dus zelf acties uitvoeren wanneer de gebruiker zijn locatie is gevonden. Dit geeft een generiek ontwerp waar het target zelf niets met locaties hoeft te doen om alsnog de huidige locatie te achterhalen. Naast het ophalen van de huidige locatie van de gebruiker was er een functionaliteit die geporteerd moest worden van de Week Calendar app naar het WeekCal framework, namelijk: Geocoding.

Geocoding is het opzoeken van coördinaten aan de hand van een locatie naam en vice versa. Het uitvoeren van deze taken op het apparaat van de gebruiker zou alleen mogelijk zijn als in de app alle map informatie van de hele wereld wordt opgeslagen, wat niet realistisch is. Om deze reden wordt het geocoden gedaan door een externe service. Veelgebruikte services zijn Apple Geocoding en Google Maps API, WeekCal had echter al een eigen API. Ik heb dus gebruik gemaakt van de al bestaande API van WeekCal waar ik slechts een POST call naar hoefde te sturen met de locatie naam waarvan de coördinaten nodig waren. Onder het kopje Networking wordt meer beschreven over hoe GET en POST requests naar API's worden gedaan.

4.2.3.3.6 Util Module

De Util module is niet zozeer een zelfstandige module maar eerder een collectie van klassen die allemaal hun eigen specifieke doel hebben en niet onder andere modules vallen.

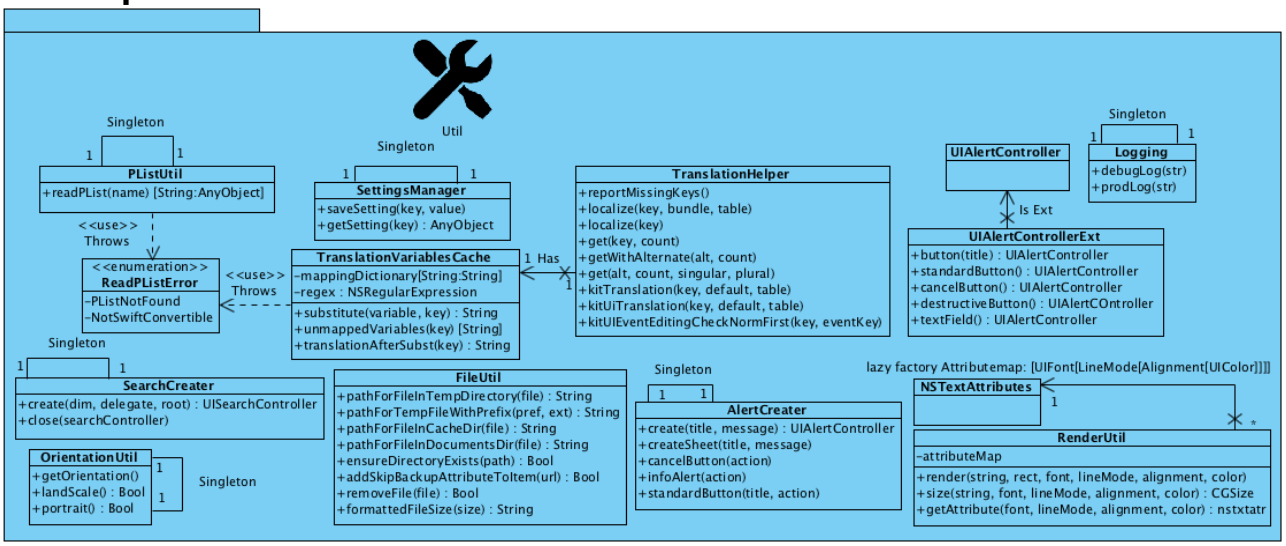
Requirements

De volgende requirements vallen onder de Util module:

Requirements Nummer	Beschrijving
R12	Het framework moet kunnen loggen daar de console in Debug en Release Mode
R13	Het framework moet gebruikers preferences managen
R26	Het framework moet strings kunnen renderen naar CoreGraphics context
R33	Het framework moet teksten kunnen vertalen (localization)
R34	Het framework moet property lists kunnen inlezen en schrijven
R35	Het framework moet rauwe bestanden kunnen inlezen en schrijven
R48	Het framework moet URL schema's kunnen registeren

Tabel 11. Requirements Util module

Ontwerp



Afbeelding 33. Ontwerp Util Module

Het ontwerp van de Util module bestaat uit veel losstaande utility klassen. De meeste van deze utility klassen maken gebruik van het singleton pattern. Hierdoor is het mogelijk om de utility klassen lazy te laden (alleen laden wanneer nodig) en een duidelijke consistente toegang te geven aan utility klassen. Naast klassen die zijn te herleiden uit de requirements komen er ook andere klassen voor in dit ontwerp, veel van deze klassen zijn ontworpen bij het moderniseren van deprecated code en dus later toegevoegd aan het ontwerp (*OrientationUtil, AlertCreater*).

Behalve het singleton pattern wordt er ook gebruik gemaakt van het lazy factory pattern. De klasse *RenderUtil* maakt *instanties NSTextAttributes* en houdt deze bij in de map *attributeMap.* Dit pattern wordt gebruikt om de performance van *RenderUtil* te waarborgen, in het oude ontwerp werd deze performance versterkende techniek niet

gebruikt. Verdere toelichting over het Lazy Factory pattern is beschreven in hoofdstuk 4.2.1.1.1.

Tot slot word er in de Util module gebruik gemaakt van nog één design pattern, namelijk het Builder pattern. In de extensie (hoofdstuk 4.2.1.1.2) klasse *UIAlertControllerExt* zijn een aantal functies gedefinieerd die allemaal het type UIAlertController retourneren. Er wordt hier dus een combinatie van het Builder pattern en de extensie techniek gebruikt wat het mogelijk maakt om het builder pattern toe te passen in een klasse van de iOS SDK.

Toelichting Module
De bovenstaande requirements zijn niet aan elkaar gekoppeld maar zijn wel allemaal belangrijk, de meest interessante worden hieronder beschreven.

Logging (R12)

Logging is een belangrijk onderdeel bij het ontwikkelen en onderhouden van een applicatie. Logging houdt in het sturen van informatie naar de console in runtime, de console is een scherm in XCode waar logging naartoe wordt gestuurd. Een ontwikkelaar kan hiermee dus runtime informatie verkrijgen.

Logging is heel makkelijk in iOS maar er is wel een probleem, namelijk dat logs alleen kunnen worden gestart met enkele regels code. Wanneer er dan een release wordt gemaakt zouden al deze regels moeten worden verwijderd of uitgezet, een taak die veel tijd kost waarbij fouten gemaakt kunnen worden. Logs die in de applicatie blijven tijdens productie kunnen fouten, crashes en verminderde performance veroorzaken, het is dus essentieel dat er geen debug logs voorkomen in de productie omgeving.

Om deze reden wordt er gebruik gemaakt van runtime flags [14]. Runtime flags zijn sleutels die kunnen worden meegegeven aan een applicatie (of framework) wanneer er een build (debug of release) wordt gemaakt. Het is in XCode mogelijk om bij bijvoorbeeld de release build de runtime flag 'PRODUCTION' mee te geven, deze runtime flag kan vervolgens worden uitgelezen in code. In afbeelding 34 staat afgebeeld hoe dit in code wordt gedaan. Als de runtime flag "*PRODUCTION*" bestaat wordt de functie DebugLog gestopt nog voordat de logging functie (*print*) kan worden uitgevoerd.

```
func DebugLog(log : String) {
    #if PRODUCTION
        return
    #endif
    print("\(NSDate()): \(log)")
}
```

Afbeelding 34. Runtime Flag Check

De production flag bestaat alleen in een release build, bij debug builds wordt de *print* functie dus wel uitgevoerd. Op deze manier is het mogelijk om zonder extra werk te loggen en is het probleem van overbodig loggen bij release builds opgelost. Om alsnog te kunnen loggen in release builds heb ik een functie gemaakt die '*ProductionLog*' heet, deze functie heeft geen runtime flag check en zal dus ook loggen in release builds.

Preferences

Preferences zijn opgeslagen stukjes informatie, hierbij kan bijvoorbeeld worden gedacht aan welk thema de gebruiker als laatst had gekozen. Er zijn drie veelgebruikte manieren om informatie op de gebruiker zijn telefoon op te slaan die allemaal hun eigen voordelen hebben.
- CoreData

CoreData is een door Apple geschreven framework dat wordt gebruikt om lokale SQL databases te maken. De voordelen van CoreData zijn: Snel, Data migratie ondersteuning. Dit zijn sterke voordelen maar ik heb er toch voor gekozen geen CoreData te gebruiken, het vereist namelijk veel code en onderhoud om effectief CoreData te implementeren [17].

- Binair

  Met binair wordt bedoeld dat er een bestand wordt geschreven met een zelf gemaakte structuur, het grootste voordeel hiervan is de controle en veiligheid. Als ontwikkelaar ben je de enige die gemakkelijk de bestanden kan veranderen, bij CoreData is dit voor derde partijen makkelijker. De preferences bij WeekCal hebben echter deze veiligheid niet nodig en zelf een formaat schrijven kost veel tijd.

- NSUserDefaults

  NSUserDefaults is een door Apple geschreven systeem dat gebruikt wordt om informatie op te slaan die snel moet kunnen worden opgezocht en geschreven [16]. NSUserDefaults biedt geen verhoogde veiligheid [15] maar heeft wel de mogelijkheid om informatie tussen extensies (Watch extension, Today extension) en de main applicatie te delen. Hiernaast is NSUserDefaults makkelijk om te implementeren [16] en zijn migraties simpel. Om deze reden heb ik gekozen om de preferences op te slaan met NSUserDefaults.

```
func saveSomething() {
    //Sla de waarde 10 op voor sleutel "sleutel_1"
    standardDefaults.setInteger(10, forKey: "sleutel_1")
    //Haal de waarde voor sleutel_1 op
    standardDefaults.integerForKey("sleutel_1")
}
```

Afbeelding 35. NSUserDefaults opslaan en ophalen

In afbeelding 35 wordt voor de sleutel 'sleutel_1' de waarde 10 gezet, deze waarde kan direct weer worden opgehaald door '*integerForKey…*' uit te voeren. Een makkelijke manier om kleine stukjes informatie op te slaan dus. Om het makkelijker te maken om grotere stukken informatie op te slaan heb ik gebruik gemaakt van het Builder pattern (hoofdstuk 4.2.1.1.1).

```
func metBuilderPattern() {
    setInt("key1", value: 10).setBool("key2", value: true).setString("key3", value: "Hallo").synchronize()
}
func geenBuilderPattern() {
    setInt("key1", value: 10)
    setBool("key2", value: true)
    setString("key3", value: "Hallo")
    synchronize()
}
```

Afbeelding 36. Builder vs Geen builder

In afbeelding 36 is te zien dat met gebruik van het builder pattern er in plaats van 4 regels slechts 1 regel code nodig is. Hoewel dit niet de originele bedoeling is van het builder pattern, komt het Builder pattern wel goed uit in dit specifieke geval.

Localization

Localization betekend het beschikbaar maken van software voor verschillende talen. In het WeekCal framework worden geen vertalingen opgenomen, dit is namelijk niet generiek. Net zoals bij de theming module wordt er bij localization slechts een structuur ontwikkeld die het mogelijk maakt gemakkelijk vertalingen toe te voegen en op te vragen in de Week Calendar app (en alle andere targets).

Binnen XCode wordt een localization gedefinieerd als een sleutel en een daarbij horende waarde, een dergelijk key-value is bijvoorbeeld: “hello_message” = “Hallo!”. Deze key kan dan worden gedefinieerd per taal en op die manier dus gelokaliseerd worden. Tot slot kan de localization worden opgevraagd met gebruik van de sleutel.

Strings renderen naar CoreGraphics context
Om strings te laten zien in een app moeten deze worden getekend naar de CoreGraphics context. iOS kan vervolgens deze context omzetten naar een visueel beeld en het laten zien aan de gebruiker. Om een goed vloeiend scherm te behouden moet dit 60 keer per seconde worden gedaan wat betekent dat er geen intensieve logica kan worden uitgevoerd want dit zal ten koste gaan van het vloeiende scherm. Dit was een probleem bij het tekenen van tekst, om tekst te tekenen moest er namelijk eerst een ‘*NSTextAttributes*’ object worden aangemaakt. iOS gebruikt dit object om te bepalen hoe de tekst wordt getekend (tekst grootte, kleur, alignment). 60 keer per seconde, per string text attributen aanmaken kost veel rekenkracht en geheugen, om deze reden heb ik hier gebruik gemaakt van het lazy factory design pattern. Bij het lazy factory design pattern worden objecten zoveel mogelijk hergebruikt waardoor zowel rekenkracht als geheugen wordt bespaard.

```
var textStyleMap = [UIFont:[NSLineBreakMode:[NSTextAlignment:[UIColor:[String:AnyObject]]]]]()
```

Afbeelding 37. Lazy Factory map

In afbeelding 37 staat de map afgebeeld waarin alle al gemaakte tekst attributen staan opgeslagen, dit is een nested map waar iedere map weer een andere map bevat. Met deze aanpak is het mogelijk om met zo min mogelijk rekenkracht de correcte tekst attributen op te halen.

```
if let savedTextStyle = textStyleMap[font]?[lineBreakMode]?[alignment]?[color] {
    return savedTextStyle
}
```

Afbeelding 38. Waarde uit map halen

4.2.3.3.7 Date Utilities Module

De Date Utilities is net zoals de Util module een collectie van klassen die, met lage koppeling, allemaal hun eigen doel hebben. Al deze klassen heb echter wel hetzelfde onderwerp: datum en tijden. Aangezien de Week Calendar app, waarvoor het WeekCal framework initieel wordt gemaakt, een agenda app is, is het niet onverwachts dat deze module veel functionaliteit heeft.

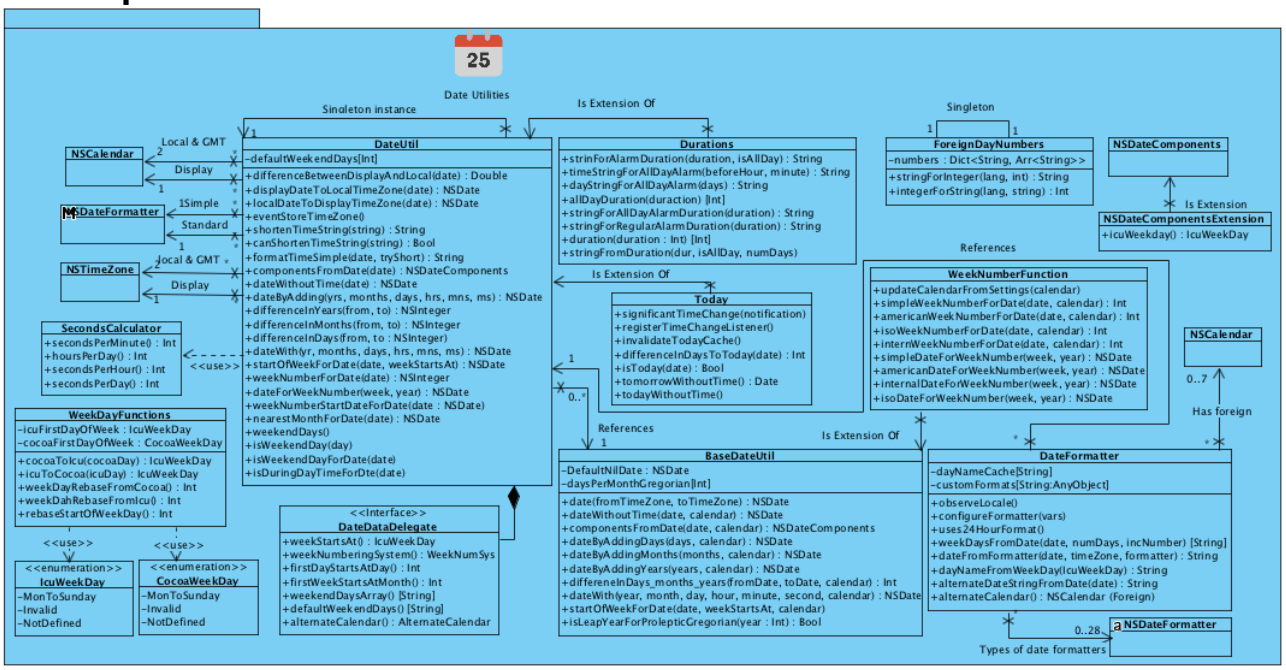
Requirements
De volgende requirements vallen onder de Date Utilities module:

Requirements Nummer	Beschrijving
R1	Het framework moet datums kunnen opmaken.
R2	Het framework moet bij het opmaken van datums meerdere calendars kunnen ondersteunen (perzisch, moslim, Hindi)
R3	Het framework moet datum formaten kunnen lezen uit property lists
R4	Het framework moet zowel GMT, display en EKEventStore tijdzones ondersteunen
R5	Het framework moet tussen tijdzones kunnen rekenen

Requirements Nummer	Beschrijving
R6	Het framework moet zowel ICU als Cocoa weekdagen ondersteunen
R7	Het framework moet zowel Europese als Amerikaanse manier van datums ondersteunen
R8	Het framework moet zowel de 12 uurs als 24 uurs klok ondersteunen
R9	Het framework moet met datums kunnen rekenen (dagen, maanden, jaren toevoegen/ verwijderen) Verschillen meten. Jaren, Maanden, dagen, weeknummers vinden
R10	Het framework moet kunnen detecteren of een datum binnen een weekend valt
R11	Het framework moet tijden kunnen omzetten naar alarmen (push notificaties)

Tabel 12. Requirements Date Utilities Module

Ontwerp



Afbeelding 39. Ontwerp Date Utilities Module

Het ontwerp van de Date Utilities module bestaat net zoals bij de Util module uit veel alleenstaande klassen, het verschil tussen de modules is dat alle klassen in dit ontwerp met datum's, tijden of tijdzones werken. De module bestaat vooral uit functies die uit de oude week calendar app moesten worden omgezet naar het WeekCal framework. Hiernaast heb ik een aantal verbeteringen in het ontwerp aangebracht waaronder het strategy pattern, extensions en Enum's.

Binnen het ontwerp zijn er twee grote klassen genaamd *DateUtil* en *BaseDateUtil,* deze klassen zijn verantwoordelijk voor de meeste datum, tijd en tijdzone bereken functies. Het god klasse probleem kwam hierdoor al snel naar voren, om dit probleem gedeeltelijk tegen te gaan heb ik specifieke functionaliteit binnen extensions ontworpen. *Durations* is bijvoorbeeld een extensie van *DateUtil,* alle tijd berekeningen worden in *Durations* gedefinieerd waardoor de grootte van *DateUtil* afneemt maar de functionaliteit hetzelfde blijft.

In het oude ontwerp was *DateUtil* niet generiek en gebruikte het dus een aantal target specifieke codes. Om dit probleem op te lossen heb ik gebruik gemaakt van het strategy pattern, *DateUtil* vraag nu het target specifieke gedrag op van

DateDataDelegate. Het target (de implementatie) kan vervolgens het specifieke gedrag uitvoeren waardoor het WeekCal framework generiek blijft.

Toelichting Module

Net zoals bij de Util module zijn de requirements van de Date Utilities module niet aan elkaar gekoppeld, hoewel het bij iedere requirement het onderwerp wel binnen datum en tijd valt. Omdat niet alle requirements in deze module complex of interessant zijn worden alleen de meest interessante hieronder beschreven.

Datums Opmaken

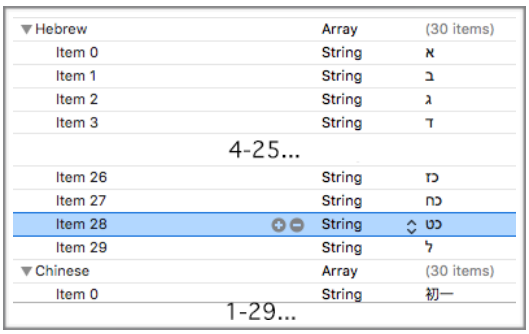
Met datums opmaken wordt bedoeld dat een NSDate object wordt omgezet naar een leesbare representatie, zoals '1 Januari, 2016'. Dit doel kan bereikt worden met gebruik van de door Apple aangeleverde '*NSDateFormatter*' en correct gebruikt van de ICU standaard (hoofdstuk 4.2.3.4.8). In het WeekCal framework project wordt er alleen gebruik gemaakt van tijd formatting en calculaties met ICU. Alle producten van Apple hebben de ICU software standaard ingebouwd en de eerder benoemde '*NSDateFormatter*' maakt er ook gebruik van, het enige dat voor het WeekCal project nu gedaan moest worden is het correct gebruiken van ICU. Een voorbeeld hiervan staat beschreven in hoofdstuk 4.2.3.4.8.

Naast de datum formatter is er nog iets anders opvallends aan de code uit afbeelding 54 (hoofdstuk 4.2.3.4.8), het woord '*lazy*' wordt gebruikt om de datum formatter te definiëren. Dit is het lazy initialization pattern waar eerder over geschreven is bij het System Design Diagram. Door het keyword '*lazy*' toe te voegen aan de definitie van een variabele weet Swift dat deze variabele pas hoeft aangemaakt te worden zodra het nodig is, dit scheelt opstart tijd en houdt code schoner. De datum formatter uit afbeelding 54 (hoofdstuk 4.2.3.4.8) is namelijk pas nodig als een datum opgemaakt moet worden op een bepaalde manier, dit is niet altijd het geval bij het opstarten van de Week Calendar app.

Meerdere kalenders ondersteunen

Naast de gregoriaanse kalender, ofwel de westerse kalender, zijn er nog een heel aantal kalenders met compleet verschillende werking. Gelukkig ondersteunt ICU alle kalenders en hoeft er in het WeekCal framework geen extra code worden geschreven om te rekenen met andere typen kalenders.

Er is echter een benodigde functionaliteit waar ICU geen ondersteuning voor heeft, namelijk weekdagen van de Chinese en Hebreeuwse kalender. In eerste instantie zou dit geen probleem moeten zijn, de week nummers zouden gemakkelijk in de code kunnen worden gedefinieerd om het probleem op te lossen en de functionaliteit toe te voegen. Er is echter één probleem, Hebreeuws is een rechts-naar-links taal. Dit betekent dat de taal, net zoals Arabisch, van rechts naar links wordt geschreven. XCode heeft hier grote problemen mee waardoor er fouten ontstaan in de Hebreeuwse tekst. Om dit probleem op te lossen heb ik ervoor gekozen de weekdagen op te slaan in een property list, in de util module had ik al een functie gebouwd om property lists in te laden waardoor deze functie daar goed op aan sloot.



Afbeelding 40. Deel dagnummer plist

In afbeelding 40 is een deel van de property list afgebeeld, de tekens die problemen gaven in XCode zijn de meest rechter tekens in de afbeelding (כז).

ICU en Cocoa

Zoals eerder beschreven maakt de Date utilities module en iOS veel gebruik van de ICU standaard, een handige en makkelijke manier om te rekenen met datums en tijd. Binnen iOS is er echter nog een variatie bij weekdagen, een inconsistentie die veel berekeningen incorrect maakt als het niet goed wordt opgevangen. Deze inconsistentie is de manier waarop Cocoa, een framework van Apple, weekdagen definieert. Het verschil tussen Cocoa en ICU weekdagen is 1 nummer, iets kleins wat grote gevolgen heeft voor berekeningen met weekdagen. In Cocoa is de eerste weekdag Zondag, waar bij ICU Maandag als de eerste dag wordt gezien.

Week dag	ICU	Cocoa
Maandag	1	2
Dinsdag	2	3
Woensdag	3	4
Donderdag	4	5
Vrijdag	5	6
Zaterdag	6	7
Zondag	7	1

Tabel 13. ICU en Cocoa weekdagen

Zoals in tabel 13 is beschreven komt geen enkele dag tussen Cocoa en ICU overeen, een vervelende inconsistentie die op treed bij bepaalde functies van iOS waar Cocoa wordt gebruikt. Om dit probleem op te lossen heb ik functies geschreven die Cocoa omzet naar ICU en vice versa, hiernaast heb ik ervoor gekozen om altijd ICU aan te houden om inconsistenties te voorkomen.

```
let cocoaRaw = cocoaWeekDay.rawValue
if let icuDay = IcuWeekDay(rawValue: cocoaWeekDay == CocoaWeekDay.Sunday ? 7 : cocoaRaw - 1) {
    return icuDay
} else {
    return IcuWeekDay.InvalidDay
}
```

Afbeelding 41. Cocoa naar ICU conversie

In afbeelding 41 wordt een gegeven cocoa dag omgezet naar een ICU dag, de conversie kan rekenkundig gedaan worden omdat de dagen altijd met 1 verschillen.

4.2.3.3.8 Networking Module

De networking module is verantwoordelijk voor alle communicatie met externe bronnen zoals API's en web services.

Requirements
De volgende requirements vallen onder de Networking module:

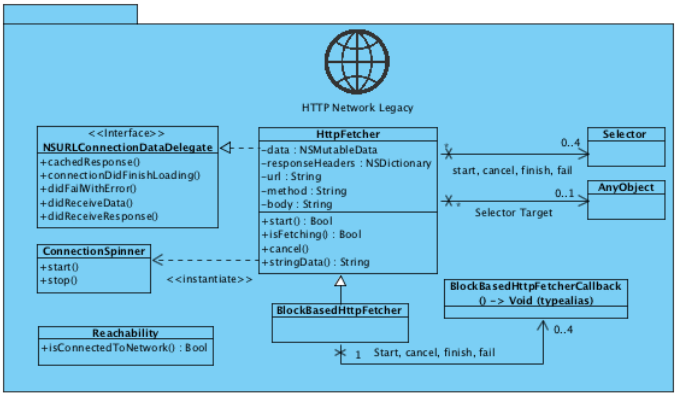
Requirements Nummer	Beschrijving
R36	Het framework moet HTTP posts en gets kunnen sturen en reageren op hun reacties

Tabel 14. Requirements Networking Module

Toelichting Module
In de Week Calendar app wordt op dit moment gebruikt gemaakt van een oude methode om GET en POST requests uit te voeren, om het WeekCal framework volledige functionaliteit te geven heb ik de oude methode geporteerd naar het framework. Deze methode is nu bekend als de Legacy Networking submodule. De moderne methode heb ik zelf geschreven, hoewel het momenteel alleen als complement wordt gebruikt voor legacy. Hiermee wordt bedoeld dat het extra functionaliteit geeft bovenop de legacy module.

Legacy

Legacy Ontwerp



Afbeelding 42. Legacy Networking Ontwerp

Zoals eerder beschreven heb ik de volledige oude methode van networking geporteerd naar het WeekCal framework, om deze reden is ook het ontwerp hetzelfde gebleven. Er worden geen design patterns gebruikt, in plaats daarvan word er voor communicatie met Selectors (hoofdstuk 4.2.3.3.10) gewerkt. In andere modules heb ik vaak Selector ontwerpen vervangen met het strategy pattern, strategy is namelijk veiliger, beter leesbaar, beter uitbreidbaar en een standaard in Swift. Om de legacy network module compatibel te houden met de huidige Week Calendar app heb ik ervoor gekozen om in dit geval het selector ontwerp te behouden.

De legacy networking module maakt gebruik van '*NSURLConnection*', dit is tot iOS 9 de standaard methode geweest om HTTP requests uit te voeren.

```
func sampleConnect() {
    let url = NSURL(string: "https://sample.com/data")!
    let request = NSMutableURLRequest(URL: url)
    request.HTTPMethod = "GET"
    let connection = NSURLConnection(request:request, delegate: self)
}
```

Afbeelding 43. Oude methode GET request

In afbeelding 43 wordt in de functie '*sampleConnect*' een HTTP GET request gestart, zodra het request voltooid is wordt de onderste functie wordt door iOS binnen het WeekCal framework uitgevoerd. Dit werkt met het al vaker gebruikte strategy pattern. In de '*sampleConnect*' functie wordt eerst de URL van de webservice gedefinieerd, vervolgens wordt een url request aangemaakt en tot slot wordt het request gestart door een *NSURLConnection* object aan te maken met het request en als delegatie de legacy network module. Door de delegatie toe te wijzen weet iOS aan wie het moet vertellen dat de request is voltooid of gefaald, wanneer het request dan voltooid is wordt *connectionDidFinishLoading* uitgevoerd door iOS in de legacy network module.

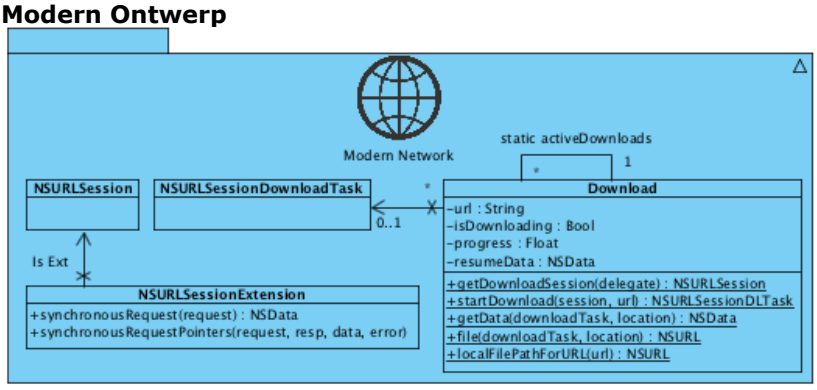
```
public func connectionDidFinishLoading(connection: NSURLConnection)
{
    guard let target = self.target else {
        return
    }
    guard let selector = self.didFinish else {
        return
    }
    if(target.respondsToSelector(selector))
    {
        target.performSelector(selector, withObject: self)
    }
}
```

Afbeelding 44. ConnectionDidFinishLoading, Selector & Guards

Binnen de '*connectionDidFinishLoading*' functie wordt eerst getest of het target en de *didFinish* selector bestaan met gebruik van guards (hoofdstuk 4.2.3.3.10). Vervolgen wordt, mits het *target* reageert op de Selector (hoofdstuk 4.2.3.3.10), de selector uitgevoerd op het *target*. Hoewel dit slechts geporteerde code is heb ik wel zelf de guards en de *respondsToSelector* check toegevoegd om de veiligheid van de code te verbeteren.

Modern

Modern Ontwerp



Afbeelding 45. Modern Networking Ontwerp

Het moderne ontwerp voegt de functionaliteit van downloads toe aan de networking module met gebruik van Apple's nieuwe klassen *NSURLSession* en *NSURLSessionDownloadTask*. De klasse *Download* houdt een lijst van concurrente downloads bij en is tegelijk zelf ook een download, er wordt met statische functies gewerkt om nieuwe downloads te kunnen starten en downloads bij te houden. Een beter ontwerp was wellicht een *DownloadManager* klasse geweest met een singleton

instantie, de manager had dan downloads kunnen managen. Door tijdgebrek heb ik echter het ontwerp op deze manier ontwikkeld.

De moderne variant van '*NSURLConnection*' maakt gebruik van de nieuwe klasse '*NSURLSession*', het voordeel van NSURLSession is dat het wordt gemanaged door iOS zelf. Dit betekent dat er geen extra code meer nodig is om meerdere requests tegelijk te kunnen versturen, hiernaast kunnen requests nu tussentijds worden gepauzeerd of gestopt. Tot slot is het veel makkelijker geworden om bestanden (zoals afbeeldingen) op te vragen van een API, NSURLSession regelt nu zelf de download buffer en het opslaan van bestanden.

4.2.3.3.9 Extensies Module

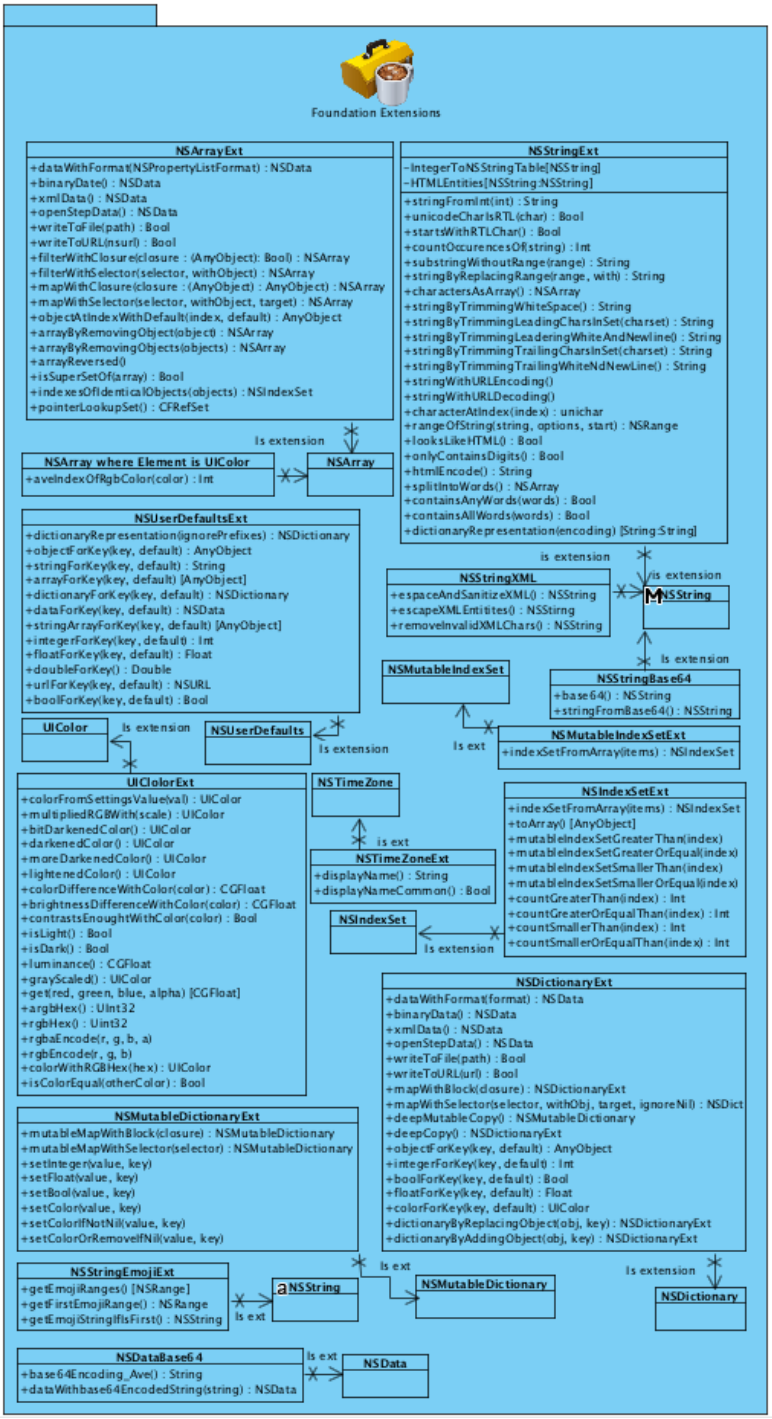
De extensies module is een verzameling van alle extensies die binnen het WeekCal framework zijn ontwikkeld. Dit is dus zoals de utility modules een module waar weinig koppeling is. De module bestaat voor het grootste gedeelte uit functies die andere modules helpen/mogelijk maken.

Requirements
De volgende requirements vallen onder de Extensies module:

Requirements Nummer	Beschrijving
R21	Het framework moet efficiënt integers naar string kunnen omzetten
R22	Het framework moet kunnen detecteren of een string Right-to-left is (arabisch)
R23	Het framework moet strings compatible kunnen met voor HTTP requests
R24	Het framework moet kunnen detecteren of een string HTML is
R25	Het framework moet strings kunnen splitten
R27	Het framework moet strings kunnen omzetten van en naar base64
R28	Het framework moet XML in strings kunnen supporten
R29	Het framework moet Emoji kunnen detecteren in Strings
R30	Het framework moet NSData van en naar base64 kunnen omzetten
R31	Het framework moet indexsets, arrays en dictionaries kunnen aanmaken, filteren en muteren

Tabel 15. Requirements Extensies Module

Ontwerp



Afbeelding 46. Ontwerp Extensions module

Het ontwerp van de extensions module bestaat geheel uit extension klassen die klassen extenden uit de iOS SDK. Op een aantal klassen na (*NSStringEmojiExt, UIColorExt*) komt dit ontwerp uit de oude Week Calendar code, om deze reden wordt er ook gebruik gemaakt van klassen als NSString in plaats van de Swift variant String. Ik heb het ontwerp zo veel mogelijk naar een Swift ontwerp omgezet door pointers weg te werken en functie namen om te zetten naar de Swift standaard.

Toelichting Module

De extensions module bestaat zoals eerder beschreven uit extension klassen die zijn afgeleid van de oude Week Calendar code. De realisatie van de module bestond dus voor een groot deel uit het omzetten van Objective-C code naar Swift code(hoofdstuk 4.2.3.4.9 & 4.2.3.4.10).

4.2.3.4 Technieken

In dit hoofdstuk worden technieken en concepten beschreven die ik heb gebruikt tijdens het ontwikkelen van de eerder beschreven modules.

4.2.3.4.1 Swift Dynamic Framework

Een Swift Dynamic Framework is een nieuwe methode om dynamische frameworks te bouwen met Swift. Het grote verschil tussen een normale library en een Swift Dynamic framework is dat een dynamic framework on-demand kan worden geladen. Klassen binnen het framework worden dus pas geladen wanneer ze worden opgevraagd [22], dit kan de opstart-tijd van de Week Calendar app versnellen. Hiernaast kan een framework (net zoals een library) aan ieder willekeurig project worden gekoppeld, alle functionaliteit en protocollen die erin staan zijn dus herbruikbaar.

4.2.3.4.2 Closures

```
/// Get all contacts from the contact store
/// - Returns: A tuple with all found contacts
public func getContacts() -> [CNContact]
{
    if(!hasAccess(true, onAccesGranted: {
        self.getContacts()
        self.delegate.contactsDelayedFetched?()
    })) {
        return [CNContact]()
    }
    /***
    * Contacten worden hier opgehaald
    ***/
    return contacts
}
public func hasAccess(shouldRequest : Bool, onAccesGranted:(() -> ())?) -> Bool
{
    let status = CNContactStore.authorizationStatusForEntityType(CNEntityType.Contacts)
    /* ... */
    if(shouldRequest && status == CNAuthorizationStatus.NotDetermined) {
        requestAccess(onAccesGranted)
    }
    return false
}
public func requestAccess(onAccesGranted:(() -> ())?)
{
    contactStore.requestAccessForEntityType(CNEntityType.Contacts) { (granted : Bool, error : NSError?) in
        runOnMainThread({
            if(granted && error == nil) {
                onAccesGranted?()
                /* ... */
            } else {
                /* ... */
            }
        })
    }
}
```

`/* ... */` = **Onbelangrijk voor dit voorbeeld**

Afbeelding 47. Closures gebruikt om toegang vragen op te lossen

In afbeelding 47 is de code weergegeven die ervoor zorgt dat het framework actie onderneemt zodra de gebruiker toegang tot zijn contacten geeft, zonder concurrency problemen te veroorzaken.

De bovenstaande code begint bij de methode *"getContacts"*, deze functie is verantwoordelijk voor het ophalen van alle contacten maar kan dit alleen doen als er toegang is gegeven. De eerste lijn van deze methode is daarom dus ook *"if(! hasAcces(true, onAccesGranted: {"*, deze lijn roept de *'hasAccess'* methode aan en geeft aan dat de volgende code moet worden uitgevoerd als er toegang is gegeven: *"self.getContacts() self.delegate.contactsDelayedFetcher?()"*, deze meegegeven code is ook meteen de closure. Vervolgens test de hasAccess methode eerst of er al

toegang is, als er nog geen toegang is wordt eerst om toegang gevraagd en vervolgens "*false*" geretourneerd. De getContacts methode wordt hierdoor afgesloten en de originele call is voltooid, dit is waar de asynchrone logica begint.

De methode requestAccess heeft eerder de gebruiker om input gevraagd en heeft ook in een closure meegegeven wat er moet gebeuren wanneer de gebruiker input heeft gegeven. Zodra de gebruiker input geeft wordt het volgende code blok uitgevoerd: "*runOnMainThread({ … onAccessGranted() })*". Omdat er via user input een functie wordt uitgevoerd is het van belang om de code uit te voeren op het main thread van de applicatie, vandaar de lijn "*runOnMainThread*". Binnen runOnMainThread staat *onAccessGranted()* ; dit is de closure die in het begin van dit voorbeeld was aangemaakt, de closure wordt dus op dit moment pas uitgevoerd.

In de closure wordt '*getContacts*' opnieuw uitgevoerd en wordt aan het delegatie (strategy pattern) verteld dat de contacten nu wel zijn opgehaald. Met closures is het dus mogelijk om synchrone en asynchrone code te koppelen en uit te voeren.

4.2.3.4.3 CocoaPods

CocoaPods is een dependency manager voor Swift en Objective-C project. Er zijn meer dan 18.000 software pakketten beschikbaar binnen CocoaPods waar iedereen aan toe kan voegen. CocoaPods werkt via de command-line en is gemakkelijk te gebruiken. Om CocoaPods toe te voegen aan een project hoeft alleen het volgende commando toe gevoegd te worden: "*pod init*", het project wordt vervolgens geconfigureerd voor CocoaPods en er wordt een podfile gegenereerd. De podfile is het bestand waarin er kan worden aangegeven welke software pakketten het project nodig heeft, een dergelijke podfile wordt in afbeelding 48 afgebeeld.

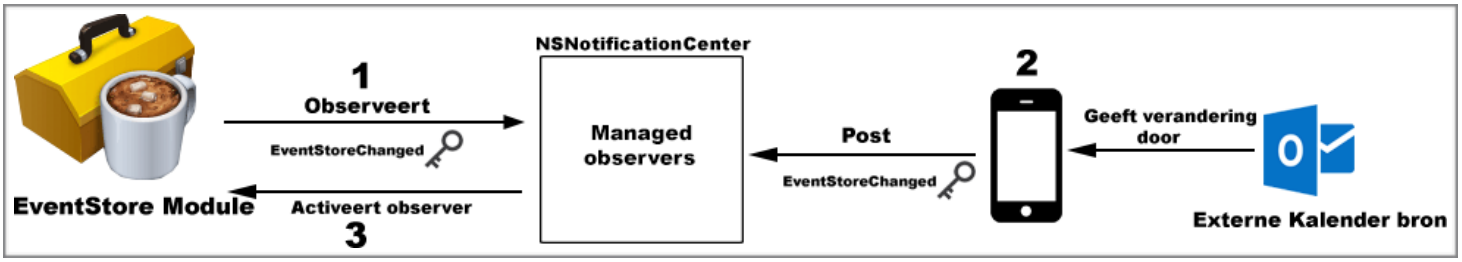
```
# Naam van target (main applicatie, watch extension, tests, frameworks)
target 'WeekCalendarFramework' do
    # pod "naam van software pakket"
    pod "EPContactsPicker"
end
```

Afbeelding 48. Podfile voor CocoaPods

In afbeelding 48 wordt voor het target: "*WeekCalendarFramework*" het software pakket "*EPContactsPicker*" als dependency gebruikt. Het is mogelijk om net zoveel software pakketten toe te voegen als dat de ontwikkelaar zelf wilt. Na het invullen van de podfile moet er via de command line het volgende commando worden uitgevoerd: "*pod install*", CocoaPods zal dan met gebruik van de podfile alle software pakketten ophalen en installeren in het gekozen target. Wanneer een software pakket niet meer volledig up-to-date is kan het commando "*pod update*" worden uitgevoerd, CocoaPods zal dan zoeken naar updates voor alle software pakketten uit de podfile en (wanneer aanwezig) deze installeren.

4.2.3.4.4 NSNotificationCenter [13]

NSNotificationCenter is een gecentraliseerde event manager voor een grote hoeveelheid processen die draaien op iOS. De manier waarop NSNotificationCenter werkt is door klassen bepaalde sleutels te laten observeren en ander processen deze te laten posten naar NSNotificationCenter als *events*. Een event bestaat uit een sleutel en optionele informatie over het event, als de sleutel van het event overeenkomt met een sleutel die wordt geobserveerd wordt de observer geactiveerd.



Afbeelding 49. Werking NSNotificationCenter

In afbeelding 49 staat de werking van NSNotificationCenter schematisch beschreven, er wordt in dit voorbeeld alleen gekeken naar de sleutel "*EventStoreChanged*". Deze sleutel wordt gebruikt wanneer er een verandering plaats vind in een van de kalenders van de gebruiker. In afbeelding 49 observeert de EventStore module de sleutel "*eventStoreChanged*" (1), er zal blijven worden geobserveerd totdat de observer expliciet wordt verwijderd. Zodra een verandering plaatsvindt in een kalender van de gebruiker zal iOS' NSNotificationCenter de observers hiervan op de hoogte brengen door "*EventStoreChanged*" te posten (2), NSNotificationCenter zoekt vervolgens naar observers die deze sleutel observeren en zal tot slot als er observers zijn gevonden deze activeren (3). Op deze manier kan code worden uitgevoerd zonder enige koppeling en dependencies tussen verschillende processen in iOS en het WeekCal framework (en iedere andere app).

```
NSNotificationCenter.defaultCenter().addObserver(self, selector:
    #selector(self.eventKitStoreUpdated), name: EKEventStoreChangedNotification, object: store)
```

Afbeelding 50. Observer toevoegen

In afbeelding 50 wordt de eerder beschreven observer toegevoegd aan het NSNotificationCenter. In deze methode is de selector een referentie naar de functie die moet worden uitgevoerd wanneer de observer wordt geactiveerd, 'name' is de sleutel en voor de parameter 'object' wordt "*store*" meegegeven zodat alleen die store wordt geobserveerd. Binnen de "*eventKitStoreUpdated*" functie worden alle kalenders opnieuw gesynchroniseerd waardoor het synchronisatie probleem wordt opgelost.

4.2.3.4.5 C Code aanroepen

De lastigste te realiseren functionaliteit van de EventStore module was het veranderen van de systeem tijdzone van het toestel van de gebruiker. In Objective-C was het mogelijk om de tijdzone te veranderen maar in Swift is deze functie verwijderd, ik moest dus een oplossing hiervoor vinden.

De oplossing die ik hiervoor heb gevonden is door met Swift te communiceren met het achterliggende C systeem, omdat de functionaliteit hiervoor wel aanwezig was in het C systeem was het mogelijk om via een omweg de systeem tijdzone te veranderen. Communiceren met C vanuit Swift is echter niet gemakkelijk en hiernaast blokkeert Apple vaak applicaties dit dit doen, hiervoor is er ook een oplossing gevonden.

```swift
public func updateSystemTimeZoneCache() -> Bool
{
    /// Fetch code bundles
    let bundle = NSBundle(forClass: EKEventStore.classForCoder())
    let path = bundle.executablePath
    guard let cPath = path?.cStringUsingEncoding(NSUTF8StringEncoding) else {
        return false
    }
    /// Fetch specific code bundle
    let handle = dlopen(cPath, RTLD_LAZY)
    if(handle != nil) {
        /// Fetch symlink name (link to symlink)
        let decodedFunctionName = aveLovesNadia("RGJtVGZ1VHp0dWZuVWpuZltwb2Y=")
        let symlinkName = decodedFunctionName.cStringUsingEncoding(NSUTF8StringEncoding)
        if(symlinkName != nil) {
            /// Fetch the symlink
            let symlink = dlsym(handle, symlinkName)
            let systemTimeZone = CFTimeZoneCopySystem()
            if(systemTimeZone != nil) {
                /// Convert the symlink to executable code
                let function = unsafeBitCast(symlink, CalSetSystemTimeZonePtr.self)
                //  Execute
                function(timeZone: systemTimeZone)
            } else {
                // Pointer to system time zone was nil (no access)
                return false
            }
        } else {
            // Pointer to symlink was nil (no access / wrong name)
            return false
        }
        // Close bundle access
        dlclose(handle)
    } else {
        // No access to the specific code bundle
        return false
    }
    return true
}
```

Afbeelding 51. C Functie Aanroepen

In afbeelding 51 staat de code weergegeven die ervoor zorgt dat de system timezone van het toestel van de gebruiker wordt geüpdatet. De functie werkt op de volgende wijze:

1. Het eerste deel van de methode haalt de code bundel op voor de klasse "*EKEventStore*" (*let bundle tot let handle*), in deze bundel word in runtime C, Obj-C en Swift code opgeslagen.
2. Uit de bundle moet vervolgens de zogenoemde SymLink voor de benodigde C functie worden gehaald (*dlsym*), een SymLink is een referentie naar een klasse, functie of variabele.
3. Om de SymLink op te halen moet eerst de naam ervan worden doorgegeven aan de *dlsym* functie, dit is waar het eerste probleem voorkomt
   - Probleem: Wanneer een app wordt verstuurd naar Apple om in de app store te zetten wordt eerst de code gescand, als er bepaalde functies in voorkomen zal de app worden afgekeurd. Een dergelijke geblokkeerde functie is degene die nodig is om de bovenstaande code te laten werken
   - Oplossing: Ik ervoor gekozen om de naam van de functie te encrypten. Dit betekent dat de naam van de functie onherkenbaar wordt gemaakt en alleen via een algoritme weer terug kan worden gevonden (*decrypt*). Om deze reden staat er in de code "*decrypt("RGJtVGZ1VHp0dWZuVWpuZltwb2Y=")*" en niet "CalSetSystemTimeZone", de decrypt functie zal echter wel "CalSetSystemTimeZone" retourneren in runtime maar zal niet worden geblokkeerd door Apple.
4. De opgehaalde SymLink wordt omgezet in een uitvoerbare functie (*unsafeBitCast*)
5. De nu uitvoerbare functie wordt uitgevoerd (*function(timeZone: systemTimeZone)*).
6. De system time zone is nu geüpdatet en de toegang naar de code bundel wordt weer gesloten (*dlclose*).

Veel van de code gebruikt *if .. != nil* statements in plaats van guards (hoofdstuk 4.2.3.4.10), dit komt doordat het hier gaat om pointers die ook nil kunnen zijn; een uitzondering op de optionals regel [25].

4.2.3.4.6 Hex waardes en UIColor's

Hex is een nummering systeem binnen computer systemen en kan gebruikt worden om kleuren te bepalen. Een hex waarde voor een kleur bestaat normaal gesproken uit een # en 6 cijfers, de eerste twee staan voor rood, de middelste twee voor groen en de laatste twee voor blauw (RGB). De code voor rood is bijvoorbeeld 0xFF0000, FF staat hier voor 255 (volledig rood) en de laatste 4 nullen zorgen ervoor dat er geen groen en blauw wordt gebruikt. Met hex waardes is het dus mogelijk om kleuren op te slaan, echter kunnen ze niet heel gemakkelijk worden omgezet naar UIColor objecten in Swift. Binnen iOS kunnen UIColor objecten weer gebruikt worden voor verdere logica met kleuren (zoals text of elementen kleur geven).

```
extension UIColor {
    ///Creates a color from given hex string
    convenience init(hexString: String) {
        //Verwijder #
        let hex = hexString.stringByTrimmingCharactersInSet(NSCharacterSet.alphanumericCharacterSet().
            invertedSet)
        var int = UInt32()
        //Zet hex om in decimaal
        NSScanner(string: hex).scanHexInt(&int)
        let a, r, g, b: UInt32
        switch hex.characters.count {
        case 3: // RGB (12-bit), low memory hex. Wordt gebruikt voor 12bit applicaties
            (a, r, g, b) = (255, (int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) * 17)
        case 6: // RGB (24-bit), de standaard rgb.
            (a, r, g, b) = (255, int >> 16, int >> 8 & 0xFF, int & 0xFF)
        case 8: // ARGB (32-bit), rgb maar met alpha (transparantie)
            (a, r, g, b) = (int >> 24, int >> 16 & 0xFF, int >> 8 & 0xFF, int & 0xFF)
        default://Onbekende hex, gebruik standaard kleur
            (a, r, g, b) = (1, 1, 1, 0)
        }
        //Apple vraagt om waardes tussen 0 en 1 voor r,g,b en a. 255 is 1 en 0 = 0, deel dus waarde door 255
            om de correcte waarde tussen 0 en 1 te krijgen.
        self.init(red: CGFloat(r) / 255, green: CGFloat(g) / 255, blue: CGFloat(b) / 255, alpha: CGFloat(a)
            / 255)
    }
}
```

Afbeelding 52. Hex naar UIColor

In afbeelding 52 staat afgebeeld hoe een hex waarde wordt omgezet naar een UIColor object. Ten eerste wordt er gebruik gemaakt van extensies zodat het UIColor type wordt uitgerust met de conversie functie. In afbeelding 52 wordt eerst de # uit de hex gehaald (‘*let hex*’) en vervolgens wordt deze omgezet naar een decimaal (‘*NSScanner*’), hierna wordt met gebruik van bitshifting (dit is een lastig te begrijpen onderwerp en wordt verder niet gebruikt in het project, om deze reden wordt dit weg gelaten uit het verslag) de rood blauw en groen waarde gehaald uit het decimaal. Tot slot worden deze gedeeld door 255 (‘*self.init*’), UIColor wil namelijk waardes tussen 0-1 en niet 0-255.

4.2.3.4.7 Eigen Base64 Encoding / Decoding

Buiten de code uit hoofdstuk 4.2.3.3.4 zijn er ook nog andere stukken code die geblokkeerd zullen worden door Apple, tenzij hier ook encryptie en decryptie voor wordt gebruikt. Om deze reden heb ik een universele decrypter gemaakt; het idee was in eerst instantie om base64 encoding te gebruiken. Dit is een open standaard die vaak gebruikt wordt om data te versturen over het internet of om bestanden op te slaan. Later bleek echter dat Apple ook de base64 encoding kon blokkeren en moest ik een nieuwe oplossing bedenken.

```
func decrypt(base64 : NSString) -> NSString
{
    let data = NSData(base64EncodedString: base64 as String, options: NSDataBase64DecodingOptions())!
    let str = NSString(bytes: data.bytes, length: data.length, encoding: NSUTF8StringEncoding)!
    let newString = NSMutableString(capacity: str.length)
    for(i) in 0..<str.length {
        var c = str.characterAtIndex(i)
        c = c - 1
        newString.appendFormat("%c", c)
    }
    return newString
}
```

Afbeelding 53. Decryptie functie

In afbeelding 53 staat de oplossing die ik voor het bovenstaande probleem heb gevonden. Er wordt nog steeds gebruik gemaakt van een base64 encoding, maar ieder character in de encoding wordt één character naar beneden gezet (*c = c - 1*). 'b' wordt bijvoorbeeld 'a', hierdoor zal een standaard base64 decoder (zoals die van Apple) geen herkenbare code meer detecteren.

Decoder	"RGJtVGZ1VHp0dWZuVWpuZltwb2Y=" wordt decoded naar
Apple	DbmTfuTztufnUjnf[pof
WeekCal Decrypt	CalSetSystemTimeZone
Standaard	DbmTfuTztufnUjnf[pof

Tabel 16. Base64 Decoders

Zoals in tabel 16 is beschreven zal alleen de WeekCal decrypt functie de correcte methode naam verkrijgen. Alle andere decoders zullen een vreemd uitziend resultaat krijgen, maar wanneer je iedere character met 1 verlaagt zal je hetzelfde resultaat krijgen als WeekCal decrypt. 'D-b-m' wordt dan dus 'C-a-l'.

4.2.3.4.8 ICU

ICU is een populair software pakket voor datum formatting, tijd calculaties, unicode en regular expressions.

```
/// Gets date as Mon, 1 Jan 1970
public lazy var shortDayAndDateFormatter : NSDateFormatter = {
    let formatter = NSDateFormatter()
    formatter.dateFormat = "EEE, d MMM yyyy"
    return formatter
}()
```

Afbeelding 54. Date Formatter aanmaken

In afbeelding 54 wordt een '*NSDateFormatter*' object aangemaakt en wordt het datum formaat '*EEE, d, MMM, yyyy*' toegekend. ICU weet aan de hand van dit formaat hoe het de datum moet opmaken, hiernaast houdt ICU ook rekening met de systeemtaal van de gebruiker zijn toestel en wordt de datum correct opgemaakt voor iedere ondersteunde systeemtaal. In de Verenigde Staten wordt bijvoorbeeld een datum geschreven als maand-dag-jaar ter
wijl in Europa het formaat dag-maand-jaar wordt gebruikt. ICU zorgt ervoor dat deze verschillen correct worden ondersteund zonder dat hier extra code voor nodig is in het WeekCal framework. Alle ICU datum formaten zijn op te zoeken in de documentatie van het ICU project[3].

4.2.3.4.9 Code omzetten van Objective-C naar Swift

Zoals eerder beschreven heb ik veel code moeten omzetten van Objective-C naar Swift, dit is een redelijk gemakkelijke taak voor diegenen die ervaren zijn in zowel Objective-C als Swift. Als er niet genoeg kennis is van beide talen worden er namelijk snel fouten gemaakt door het misinterpreten van code. Hoewel Swift en Objective-C dezelfde Apple frameworks gebruiken (en dus dezelfde namen gebruiken) verschilt de syntax van beide talen wel sterk, dit komt doordat Objective-C is gebaseerd op C en ontwikkeld is in 1984 terwijl Swift is ontwikkeld in 2014 en gebaseerd is op de syntax van talen als Java en C#.

```swift
public func numEventAvailabilityStatusesSupported() -> Int
{
    guard supportedEventAvailabilities == .None else {
        return 0
    }
    var num = 0
    let countedEventTypes : [EKCalendarEventAvailabilityMask] = [.Busy, .Free, .Tentative, .Unavailable]
    for(type) in countedEventTypes {
        if(supportedEventAvailabilities.contains(type)) {
            num += 1
        }
    }
    return num
}
```

Afbeelding 55. Swift code

```objc
-(NSInteger)aveNumEventAvailabilityStatusesSupported
{
        if(self.supportedEventAvailabilities == EKCalendarEventAvailabilityNone)
                return 0;

        NSInteger num = 0;
        if((self.supportedEventAvailabilities & EKCalendarEventAvailabilityBusy) != 0) num++;
        if((self.supportedEventAvailabilities & EKCalendarEventAvailabilityFree) != 0) num++;
        if((self.supportedEventAvailabilities & EKCalendarEventAvailabilityTentative) != 0) num++;
        if((self.supportedEventAvailabilities & EKCalendarEventAvailabilityUnavailable) != 0) num++;

        return num;
}
```

Afbeelding 56. Objective-C code

De code in afbeelding 55 en 56 hebben dezelfde functionaliteit maar een verschillende syntax en manier van werken. De code in afbeelding 55 is de Swift methode, waarin technieken zoals guards, tuples, Swift enums en verbeterde bitshifting wordt gebruikt. De code in afbeelding 56 is de oude Week Calendar manier waarop de code in Objective-C is geschreven. Het is dus duidelijk dat behalve het veranderen van de syntax er bij het omzetten van Objective-C naar Swift nog veel meer factoren meespelen, met name het gebruiken van Swift standaarden en technieken.

Letterlijk overgenomen Objective-C code is in Swift vaak lelijk of incorrect. De code in deze afbeelding is slechts een van de vele situaties waar er andere oplossingen mogelijk zijn in Swift die in Objective niet bestaan of niet efficiënt zijn.

4.2.3.4.10 Gebruikte Swift technieken

In dit hoofdstuk worden belangrijke Swift technieken beschreven die ik heb gebruikt tijdens de ontwikkeling van het WeekCal framework.

Optionals

In Swift is een nieuw concept gebracht genaamd optionals. Een optional is een variabele die ofwel een waarde kan hebben of *nil* is, niet optionals kunnen binnen Swift nooit *nil* zijn. Om aan te geven dat een variabele optional is wordt in de declaratie van de variabele een vraagteken toegevoegd.

'var optionalString : String?'

Hierdoor weet de compiler wanneer een variabele wel of niet optional is en zal de compiler een foutmelding geven wanneer er fout gebruik van wordt gemaakt. Fout gebruik is bijvoorbeeld een optional gebruiken waar een niet optional nodig is, of een non optional een *nil* waarde geven.

Een optional is als het ware een rapper voor een echte waarde, om de echte waarde van een optional te gebruiken moet deze eerst unwrapped worden. Unwrappen van optionals kan op een aantal manieren worden gedaan:

Standaard Unwrapping

De standaard manier om een optional te unwrappen is met de volgende code:

‘*if let string = optionalString {*
optionalString is nu unwrapped in string.
}’

Door de bovenstaande code uit te voeren kan een code blok worden uitgevoerd met als conditie dat optionalString niet *nil* is, de variabele *let string* heeft vervolgens de unwrapped waarde van *optionalString* verkregen.

Met deze techniek is het dus niet langer nodig om bijvoorbeeld code als *‘if optionalString != nil’* uit te voeren en de waarde is direct unwrapped, integendeel tot de laatste code waar de optional alsnog force unwrapped moet worden. Een nadeel van deze techniek is dat bij veel optionals het pyramid of doom probleem kan voorkomen.

‘*if let string = optionalString {*
if let color = optionalColor {
if let string2 = optionalString2 {
}
}
}’

Bij drie optionals begint het probleem al, gelukkig is binnen Swift hier een oplossing voor: Guards

Guards

Guards zijn een speciale manier om optionals te unwrappen zonder het pyramid of doom probleem te veroorzaken [18]. Het pyramid of doom probleem voorbeeld wordt hieronder opgelost met guards:

‘*guard let string = optionalString else { return }*
guard let color = optionalColor else { return }
guard let string2 = optionalString2 else { return }’

In het bovenstaande voorbeeld wordt het zelfde bereikt als het eerdere voorbeeld alleen is er ditmaal geen pyramid of doom ontstaan. In het WeekCal framework wordt vaak een combinatie van beiden methodes gebruikt:

Met Guards

```
public func start() -> Bool
{
    guard let url = self.url else {
        return false
    }
    guard let method = self.method else {
        return false
    }
    if(connection != nil) {
        return false
    }
    if let nsURL = NSURL(string: url as String) {
        let urlRequest = NSMutableURLRequest(URL: nsURL)
        /** Verdere code **/
    }
}
```

Zonder Guards

```
public func start() -> Bool
{
    if let url = self.url {
        if let method = self.method {
            if(connection != nil) {
                return false
            }
            if let nsURL = NSURL(string: url as String) {
                let urlRequest = NSMutableURLRequest(URL: nsURL)
                /** Verdere code **/
            }
        }
    }
}
```

Afbeelding 57. Guards VS geen Guards

Zoals te zien is in afbeelding 57 is het voorbeeld met guards leesbaarder dan zonder guards, terwijl de functionaliteit van de code exact hetzelfde blijft.

Forced Unwrapping

Forced unwrapping is een manier om optionals te unwrappen zonder eerst te testen of ene optional wel een waarde heeft. Als de optional geen waarde heeft en force unwrapped wordt, zal de applicatie crashen. Om deze reden mag er alleen gebruik gemaakt worden van forced unwrapping wanneer het zeker is dat de opional niet *nil* is. In de praktijk moet forced unwrapping zoveel mogelijk worden vermeden [19] door gebruik van de eerder beschreven methodes. Om te force unwrappen wordt een uitroepteken achter de optional gezet:

‘*var string : String? = “Hallo”*
print(string!)’

Swift Enums
Swift Enum's worden gebruikt om staten of typen te declareren voor een bepaald doeleinde. In het WeekCal framework wordt er bijvoorbeeld voor ICU weekdagen een enum gebruikt.

```
public enum IcuWeekDay : NSInteger
{
    case InvalidDay = -1
    case NotDefined = 0
    case Monday = 1
    case Tuesday = 2
    case Wednesday = 3
    case Thursday = 4
    case Friday = 5
    case Saturday = 6
    case Sunday = 7
}
```

Afbeelding 58. Swift Enum ICU

Met enums is het mogelijk om zonder grote hoeveelheid constanten met unieke waardes te declareren alsnog unieke constanten te hebben. Met gebruik van Swift enums wordt het magic numbers (*waardes zonder label of betekenis zwervend in code*) probleem opgelost, er wordt nu namelijk gerefereerd naar de enum: '*let maandag = IcuWeekDay.Monday*'
Dit is veel beter dan bijvoorbeeld '*let maandag = 1*' en geeft ook de mogelijkheid tot uitbreiden en een generiek ontwerp.

Tuples
Tuples zijn de vervanging voor de oude *NSArray* en *NSDictionary* klassen uit Objective-C, tuples zijn makkelijker te gebruiken dan *NSArray* en *NSDictionary* en kosten ook een stuk minder regels. Tuples zijn in feite dus gewoon Arrays en Mappen met een verbeterde bruikbaarheid, een tuple aanmaken kost slechts één regel code: '*let namen = ["Guus", "Maarten", "Maurice"]*'.

Binnen het WeekCal framework worden tuples vaak gebruikt als return waarde van functies die meerder waardes moet retourneren. In Objective-C werd er in z'n geval vaak gebruik gemaakt van pointers, iets wat nog altijd kan in Swift. Het is echter voor de stabiliteit en veiligheid van het WeekCal framework beter om zo min mogelijk met pointers te werken, om deze reden gebruik ik tuples [20] als oplossing wanneer meerdere waardes geretourneerd moeten worden:

```
public func duration(duration: Int) -> [Int]
{
    var totalDuration = duration
    let days = totalDuration / (3600 * 24)
    totalDuration -= days * (3600 * 24)
    let hours = totalDuration / 3600
    totalDuration -= hours * 3600
    let minutes = (totalDuration) / 60
    totalDuration -= minutes * 60
    let seconds = totalDuration
    return [seconds, minutes, hours, days]
}
```

Afbeelding 59. Tuple als return waarde

In afbeelding 59 wordt een array geretourneerd met daarin de berekende secondes, minuten, uren en dagen. Op de oude manier zouden er pointers worden meegegeven aan de *duration* functie die vervolgens binnen de *duration* functie een waarde krijgen toegewezen. Naast dat de oude methode onveilig en instabiel is, is de nieuwe methode ook een stuk leesbaarder.

Selector
Een selector is een referentie naar een functie die, mits correct toegepast, kan worden uitgevoerd. Binnen Swift zijn Selectors voor het grootste gedeelte overgenomen door

closures omdat met *closures* hetzelfde kan worden bereikt op een leesbaardere manier.

Selectors zijn echter nog altijd een krachtige techniek die in het WeekCal framework op bepaalde locaties nog altijd wordt gebruikt (voor legacy redenen).

```
public func connectionDidFinishLoading(connection: NSURLConnection)
{
    guard let target = self.target else {
        return
    }
    guard let selector = self.didFinish else {
        return
    }
    if(target.respondsToSelector(selector))
    {
        target.performSelector(selector, withObject: self)
    }
}
```

Afbeelding 60. ConnectionDidFinishLoading, Selector & Guards

In afbeelding 60 (uit legacy networking) worden nog altijd selectors gebruikt als callback. De code die de legacy networking module gebruikt kan een selector meegeven en zichzelf als target markeren. De selector refereert dan naar een methode die buiten het WeekCal framework kan liggen maar toch vanuit het framework kan worden uitgevoerd met gebruik van de Selector (*didFinish*).

Typealias
Een typealias is zoals de naam doet denken een techniek waarmee het mogelijk is om een bepaald type een andere naam te geven. Bij het gebruiken van closures kan in Swift de code lelijk en moeilijk leesbaar worden [21], een bepaalde closure kan er als volgt uit komen te zien:

```
func findCurrentLocation(callback: (CLLocation?, NSError?) -> ([String]))
```

Dit probleem kan gelukkig makkelijk worden opgelost met een typealias.

```
typealias CurrentLocationCallback = (CLLocation?, NSError?) -> ([String])
```

Door simpelweg de bovenstaande code te definiëren kan *'CurrentLocationCallback'* in plaats van *'((CLLocation?, NSError?) -> ([String]))'* overal worden gebruikt:

```
func findCurrentLocation(callback: CurrentLocationCallback)
```

4.2.3.5 Unit Testing

Alle modules maken samen de volledige functionaliteit van het WeekCal framework. Alle requirements uit het requirements rapport zijn voltooid in deze modules op één requirement na, Namelijk R47 'Alle belangrijke functies moeten unit-tested zijn'.

Omdat ik tijdens het project heb gewerkt met de systeem ontwikkelingsmethode Test-Driven Development werd deze requirement constant uitgevoerd. Voor iedere functie heb ik eerst een unit-test geschreven met de verwachte output en daarna pas de echte functionaliteit, vervolgens kon ik de unit-test uitvoeren om te valideren of de functionaliteit correct werkt. Indien de unit-test niet slaagde werd de implementatie van de functionaliteit aangepast totdat de unit-test wel slaagde.

Deze aanpak heeft ook het probleem 'wat moet er getest worden' voorkomen omdat alles getest moest worden, wel zat er verschil tussen functies die vaker uitgevoerd worden en functies die nauwelijks of nooit gebruikt worden. Functies die vaak gebruikt worden en dus een hogere risicofactor hebben, hebben meer geteste paden; hiermee wordt bedoeld dat alle (of zoveel mogelijk) mogelijke uitkomsten van de functionaliteit worden getest.

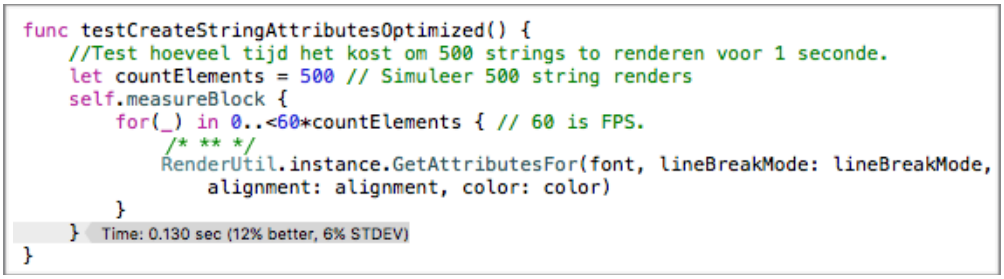
Ik heb de volgende testontwerp-technieken gebruikt:

- syntax test
  Gebruikt om functies met user-input te testen, dit kwam veel voor in het WeekCal framework
- Equivalentieklassen
  Veel functies in het WeekCal framework gedragen zich verschillend bij verschillende tijd eenheden (seconde, minuut, uur, dag). Met equivalentieklassen tests worden deze gevallen opgevangen. Grenswaarden tests maken de functies vervolgens volledig testbaar.
- Grenswaarden test
  In het WeekCal framework wordt er veel met datums, weken, maanden, tijden en dagen gewerkt. Hierdoor zijn er veel functies die goed moeten letten op grens waarden, denk hierbij bijvoorbeeld aan wanneer een maand begint en eindigt. Grenswaarden tests waren hiervoor geschikt.

Ook worden er niet functionele eisen getest met performance tests. De unit tests voor het WeekCal framework heb ik gemaakt met behulp van XCTest, het unit-test framework dat door Apple wordt aangeleverd bij XCode. XCTest maakt samenhang tussen tests en de geteste code zo sterk dat XCode zelfs exact bijhoudt welke code er is getest en laat dit ook zien in de code editor, dit heet code-coverage tracking.

Afbeelding 61. Code-coverage in code editor

In afbeelding 61 staat de XCode code editor afgebeeld met code-coverage tracking, de code in het groene vlak is hier bij de laatste testrun 336.000 keer uitgevoerd. Dit aantal is zeer hoog omdat er performance tests zijn gemaakt voor deze functie die de functie duizenden keren uitvoert om een gemiddeld tijdsbestek te verkrijgen. De test die de bovenstaande code-coverage heeft veroorzaakt is een performance test, dit is een iets andere variatie unit test die lastiger werkt. De test zier er als volgt uit.



Afbeelding 62. Performance test

In afbeelding 62 wordt de performance van de functie '*GetAttributesFor*' getest, XCode begint tijd bij te houden vanaf '*measureBlock*' tot het einde van dat code blok. Vervolgens laat XCode zien hoe lang het code blok erover gedaan heeft, in het geval van deze test dus 0.130 seconden, ofwel 130 milliseconden.

Normale unit tests werken op een iets andere manier, er wordt geen tijd gemeten maar wel wat de output is van de geteste code. Deze output wordt vervolgens vergeleken met de verwachte output, de test slaagt dan wanneer deze overeenkomen en faalt als dit niet het geval is.

```
386     func testWeekNumberForDate() {
387         let date = dateUtil.dateWithYear(2016, month: 2, day: 15, hour: 0, minute: 0, second: 0,
                useDisplayTimeZone: true)
388         let result = dateUtil.weekNumberForDate(date)
389         let expected = 7
390         XCTAssert(expected == result, "Unexpected result \(result)")
391     }
```

Afbeelding 63. Unit test weeknummer

In afbeelding 63 wordt eerst de te testen functie uitgevoerd (*let date & let result*), vervolgens wordt de verwachte output gedefinieerd (*let expected = 7*) en tot slot wordt het resultaat vergeleken met de verwachting (*XCTAssert*).

4.2.4 Deprecation Modernisation Rapport

Zoals eerder beschreven is het updaten van verouderde code (deprecated) een groot onderdeel van het WeekCal project. Binnen de Week Calendar app waren op het begin van het project circa 650 locaties met verouderde code met 25 verschillende type deprecaties. Een groot deel van deze deprecaties was het verouderde AddressBook framework, op 385 verschillende locaties werd AddressBook gebruikt.

Om een goed idee te krijgen van welke deprecated code ik allemaal heb gemoderniseerd heb ik hiervoor een document opgesteld met alle type deprecaties, de vernieuwde methode en hoe vaak de deprecatie voorkwam. Dit was geen verplicht document maar geeft veel toegevoegde waarde aan het project en is tevens een beknopte handleiding voor het implementeren van het WeekCal framework in de Week Calendar app.

Nr	Deprecated Methode	Moderne Methode	Notities	Framework implemenatie	Hoeveelheid
1	UIAlert	UIAlertController.Alert	UIAlertController heeft support voor app tint's en popover.	AlertCreater	48
2	UIActionSheet	UIAlertController.ActionSheet	UIAlertController heeft support voor app tint's en popover.	AlertCreater	6
3	device.interfaceOrientation	statusBarOrientation	StatusBarOrientation haalt de oriëntatie van de status bar en niet die van het device. Dit is dus betrouwbaarder. (interface kan portrait zijn terwijl device landscape is)	OrientationUtil	27
4	NSString.drawInRect	String.drawInRect(attributes)	De nieuwe methode heeft support voor decorated strings en is sneller	RenderUtil	47

Afbeelding 64. Onderdeel Deprecation Modernisation rapport

In afbeelding 64 staan vier deprecatie types schematisch afgebeeld, het is met dit overzicht gemakkelijk om de gemoderniseerde functie toe te passen in de Week Calendar app. De gemoderniseerde code is zover mogelijk altijd geïmplementeerd binnen het WeekCal framework.

4.2.5 Test Rapportages

Zoals eerder beschreven heb ik tijdens het project gewerkt met Test-Driven Development en unit-tests, waar alle functies worden getest. Om deze reden zullen test rapportages zeer voorspelbaar zijn en voor mij geen nuttige informatie opleveren. Voor alle andere relevante personen binnen het WeekCal framework zijn de reportages echter wel waardevol, het is direct duidelijk wat er wel en niet getest is en of alle functionaliteit naar behoren werkt. Om deze reden heb ik dus alsnog test rapportages opgesteld waarin ik heb beschreven wat en hoe er getest is. Ik heb een schema bijgehouden van één testrapportage per maand vanaf de 3e maand van het project, in het tweede rapport heb ik ook een voortijdig advies gegeven over het in productie brengen van het WeekCal framework.

4.2.6 Documentatie WeekCal Framework

Voor een correct gebruik van het WeekCal framework is het belangrijk dat diegene die het gebruiken het ook goed begrijpen. Hiervoor heb ik eerder al documentatie opgesteld zoals het requirements rapport, system design diagram en deprecation modernisation report. Deze documenten helpen toekomstige gebruikers snel een overzicht van functionaliteit en verantwoordelijkheden van het WeekCal framework te verkrijgen.

Naast deze documentatie is ook de code binnen het WeekCal framework gedocumenteerd, dit betekend dat iedere functie is uitgelegd. Hiervoor heb ik de Swift syntax gebruikt waarbij het mogelijk is om een unieke uitleg te schrijven voor de methode zelf, de parameters van methode en de return waarde. Een dergelijke functie documentatie ziet er als volgt uit:

```
/**
 Get first Contact that conforms to the given closure

 - parameter toCheck: Should this contact be returned?

 - returns: First found contact
 */
public func getFirstContact(toCheck : (CNContact) -> Bool) -> CNContact?
```

Afbeelding 65. Documentatie methode

In afbeelding 65 wordt de methode '*getFirstContact*' gedocumenteerd, de documentatie start bij '*/**' en eindigt bij '*/'. Alles hiertussen worden gedetecteerd als documentatie. XCode kan vervolgens een nette en goed leesbare documentatie weergeven met de bovenstaande geschreven documentatie, XCode detecteert namelijk woorden zoals '*parameter*' en '*returns*'. In afbeelding 66 staat afgebeeld hoe XCode de documentatie uit afbeelding 65 laat zien.

Afbeelding 66. XCode documentatie

In het plan van aanpak had ik gepland om ook een handleiding te schrijven voor het WeekCal framework. Echter heeft mijn mentor, die tevens de iOS lead developer en dus gebruiker van het WeekCal framework is, mij verteld geen behoefte te hebben aan een handleiding. Hij heeft mij geadviseerd mijn tijd te besteden aan het implementeren van het WeekCal framework in plaats van aan deze handleiding. Dit advies heb ik opgenomen, om deze reden heb ik dus geen beknopte handleiding geschreven voor het WeekCal framework.

4.2.7 Versiebeheer

Om ervoor te zorgen dat code nooit verloren gaat heb ik gebruik gemaakt van de versiebeheer tool GitLab. In eerste instantie wou ik zelf gebruik maken van GitHub maar omdat WeekCal gebruik maakt van GitLab was dat een betere optie voor compatibiliteit. Ik heb tijdens het project een schema aangehouden waarbij ik iedere dag een commit deed, tenzij ik die dag geen code had geschreven. Bij iedere commit moest een betekenisvolle beschrijving staan zodat andere werknemers van WeekCal gemakkelijk kunnen navigeren door de WeekCal framework repository.

XCode heeft een eigen version management tool waarin het bijhoudt wat de programmeur verandert in vergelijking met wat er in de repository staat. Hiernaast heeft XCode ook support voor externe git-based repositories (zoals GitLab & GitHub), hier heb ik gebruik van gemaakt door XCode te koppelen aan de GitLab WeekCal framework repository. Ik kon vervolgens met slechts een aantal knoppen een commit versturen, zonder dat ik dit moest doen via bijvoorbeeld command-line tools.

4.3 Framework Implementatie

De laatste fase van het WeekCal framework project was de framework implementatie fase, in deze fase wordt het WeekCal framework geïmplementeerd in de Week Calendar app. De Week Calendar app is geschreven in de programmeertaal Objective-C en de implementatie van het WeekCal framework moest dus ook worden geschreven in Objective-C. Dit bracht een aantal moeilijkheden met zich mee die later in dit hoofdstuk worden beschreven. Het eindproduct van deze fase is de Week Calendar app werkend met het WeekCal framework.

Zoals eerder beschreven in de opdrachtomschrijving zijn er twee Week Calendar applicaties, namelijk de iPhone en iPad applicatie. In deze fase wordt eerst het WeekCal framework geïmplementeerd in de iPhone applicatie en vervolgens, mits er nog genoeg tijd is, in de iPad applicatie.

4.3.1 Modules implementatie

De implementatie van het WeekCal framework heb ik per module uitgevoerd. Door de implementatie per module uit te voeren kon ik het werk overzien en problemen snel oplossen en dezelfde problemen voorkomen bij andere modules. Onvoorziene problemen kwamen echter al snel voor, bij de eerste module kwam ik al een aantal problemen tegen die lastig waren om op te lossen. Dit waren problemen die niet module specifiek waren en dus een goede oplossing nodig hadden, hieronder worden de problemen en hun oplossing verder uitgewerkt.

4.3.1.1 Problemen & Oplossingen

- Swift code niet beschikbaar voor Objective-C

Het WeekCal framework is in Swift geschreven en de Week Calendar app in Objective-C, twee verschillende programmeertalen die dezelfde frameworks van Apple gebruiken. Apple heeft een systeem ontwikkeld waarmee Objective-C Swift code kan gebruiken en vice versa, dit heet bridging; een heel handig systeem. Toen ik het WeekCal framework ontwikkelde had ik de gedachte dat al mijn Swift code beschikbaar zou zijn voor Objective-C via het bridging systeem, dit was echter niet het geval. Bridging van Swift naar Objective-C heeft namelijk een aantal limieten.

- Optionele bool's en int's zijn niet beschikbaar, bool's en int's kunnen dus niet null zijn.
- Default parameters zijn niet beschikbaar, in Swift is het mogelijk om bij een methode een parameter een standaard waarde te geven.
- Error handling is niet beschikbaar
- Code is alleen beschikbaar als het behoort tot een klasse die een subklasse is van NSObject; globale functies zijn dus niet beschikbaar

- Variadic parameters zijn niet beschikbaar, in Swift is het mogelijk om de laatste parameter in een methode variadic te maken; het is dan mogelijk om oneindig waardes van een bepaald type mee te geven aan de methode. De methode kan deze waardes dan als array uitlezen.
- Alleen publieke functies en variabelen zijn beschikbaar
- Swift Enums zijn niet beschikbaar

Alle bovenstaande programmeer technieken heb ik gebruikt binnen het WeekCal framework en kon ik dus niet meer gebruiken in de Week Calendar app. Ik heb dit probleem moeten oplossen door aparte functies te schrijven voor Objective-C, deze functies werden vervolgens een soort tussenlaag. De functies hadden zelf alleen functionaliteit die de niet ondersteunde methodes omzet in wel ondersteunde methodes en dus geen extra functionaliteit. Met deze aanpak kon ik nog altijd de Swift functies gebruiken binnen het WeekCal framework en toch ondersteuning leveren voor Objective-C.

```
public func getProduct(identifier : String) throws -> SKProduct?
{
    if(fetchingProducts) {
        throw GetProductError.ProductsCurrentlyLoading
    }
    /* ** */
}
@objc
public func getProductObjC(identifier : String) -> SKProduct?
{
    do {
        return try getProduct(identifier)
    } catch {
        return nil
    }
}
```

Afbeelding 67. Oplossing voor bridging

In afbeelding 67 wordt eerst de functie '*getProduct*' afgebeeld, deze functie maakt gebruik van error handling (*throw*) en is daardoor niet meer beschikbaar in Objective-C. Om het probleem op te lossen is er nog een functie gemaakt '*getProductObjC*' die de voorgaande functie gebruikt maar intern de error handling regelt. Hierdoor wordt het '*throws*' keyword uit de functie naam gehaald en is de functie beschikbaar voor Objective-C. Dit gaat dus wel ten koste van de error handling maar is de enige manier, error handling is namelijk niet beschikbaar in Objective-C en dus nutteloos om te bridgen. Deze manier heb ik ook toegepast bij de andere soortgelijke problemen die ik eerder heb beschreven.

- **Manual memory management crashes**

In Objective-C moet de programmeur zelf de objecten die hij of zij maakt weer verwijderen wanneer ze niet meer nodig zijn, dit heet manual memory management. Omdat de Week Calendar app in Objective-C is geschreven wordt er overal binnen de code rekening gehouden met memory management, wat bij de integratie van het WeekCal framework een lastig probleem heeft opgeleverd.

In Swift is manual memory management verwijderd en worden objecten automatisch verwijderd wanneer deze nergens meer worden gebruikt, dit scheelt veel code, denkwerk en fouten. Er is nu echter wel een probleem ontstaan waar ik eerder niet aan had gedacht, objecten die worden aangemaakt binnen een Swift context zullen zich anders gedragen dan objecten die zijn aangemaakt in een Objective-C context. Het heeft mij minstens een dag en heel veel debugging gekost om hier achter te komen, objecten die zijn aangemaakt in Swift en gebruikt worden binnen Objective-C worden automatisch verwijderd wanneer ze niet meer nodig zijn. Als binnen Objective-C een dergelijk object vervolgens wordt verwijderd zal het dubbel worden verwijderd, eerst wordt het object zelf verwijderd en vervolgens probeert iOS het

geheugen waar een het object stond te verwijderen. Dit betekent dat door de dubbele verwijdering er iets wordt verwijderd waar eerst het object stond, in bijna alle gevallen leidt dit ertoe dat de app crasht met een 'BAD_ACCESS of memory corruptie foutmelding. Er staat echter nooit binnen deze foutmelding welk object dubbel is verwijderd wat het erg lastig maakt om de bron van deze fouten te achterhalen en repareren.

Nadat ik achter dit probleem was gekomen heb ik veel integratie code moeten repareren waar er objecten dubbel verwijderd werden, dit is gelukkig goed gekomen en ik heb er nieuwe kennis mee opgedaan.

4.3.1.2 Implementatie

Met de kennis over de bovenstaande problemen heb ik alle modules snel kunnen implementeren. Het implementeren per module heb ik telkens op eenzelfde manier gedaan:

1. Calls van oude code omzetten naar WeekCal framework calls.

   1.1 In een aantal gevallen bestond dit simpelweg uit het importeren van het WeekCal framework in de klasse:

```
#import <WeekCalendarFramework/WeekCalendarFramework-Swift.h>
```

   Omdat ik de namen van veel functies intact heb gelaten was het in z'n dergelijk niet nodig om de call te veranderen. In sommige gevallen moest de naam van de functie wel worden veranderd omdat deze in het WeekCal framework een Swift naam had gekregen. Dit kostte nog altijd niet veel tijd.

   1.2 In de grootste hoeveelheid gevallen was de nieuwe module in het WeekCal framework ontwikkeld als een manager in combinatie met het strategy pattern (*Contacts, EventStore, IAP, …).* Hiervoor heb ik per module in de Week Calendar app een *singleton* klasse gemaakt die een object bijhoudt van de manager (en andere WeekCal framework objecten) en de strategy pattern gedragsklasse implementeert.

```
@interface Contacts : NSObject
<ContactsWrapperDelegate, ContactsHelperDelegate>

+ (Contacts*)instance;


@property (nonatomic, retain) ContactsHelper* helper;

@property (nonatomic, retain) LinkedPersonManager* linkedHelper;

@property (nonatomic, retain) ContactStoreWrapper* contactsWrapper;

@end
```

Afbeelding 68. Module implementatie in Week Calendar app (header)

In afbeelding 68 staat de header afgebeeld van de Contacts module implementatie in de Week Calendar app. De klasse houdt een object bij van iedere manager / helper klasse uit de Contacts module en een singleton instance naar zichzelf (*instance*). Hiernaast implementeert de header ook *ContactsWrapperDelegate* en *ContactsHelperDelegate* die doormiddel van het strategy pattern in het WeekCal framework waren gedefinieerd.

De bovenstaande afbeelding is zoals beschreven een header waarvan dus ook een implementatie moet worden geschreven.

```
@implementation Contacts
    Singleton instance aanmaken
+ (id)instance {
    static Contacts *instance = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        instance = [[self alloc] init];
    });
    return instance;
}
    Managers initialiseren
- (id)init {
    if (self = [super init]) {
        _contactsWrapper = [[ContactStoreWrapper alloc] initWithDelegate: self];
        _helper = [[ContactsHelper alloc] initWithWrapper:_contactsWrapper contactPickerDelegate:NULL];
        _linkedHelper = [[LinkedPersonManager alloc] initWithHelper:_helper];
    }
    return self;
}
    Gedrag implementatie
-(void) defaultBirthdayAlertChanged
{
    [[AveBadgeUpdater sharedInstance] startPendingNotification:nil];
}
```

Afbeelding 69. Implementatie Contacts header

In afbeelding 69 staat de implementatie van de header uit afbeelding 68 afgebeeld. In de implementatie wordt het singleton object aangemaakt, de managers geinitialiseerd en de gedrag implementaties geschreven. Deze aanpak heb ik ook gebruikt bij de *Theming*, *Date Utilities*, *EventStore*, *StoreKit* en een deel van de *Util* module. De andere modules en delen vielen onder de eerder beschreven aanpak (1.1).

2. Nieuwe implementatie testen

Eerst heb ik na iedere implementatie een testrun gedaan over het WeekCal framework om er zeker van te zijn dat er geen problemen waren ontstaan in het WeekCal framework. Vervolgens heb ik handmatig de Week Calendar app getest met de implementatie, ik heb hier geen specifieke techniek of methode voor gebruikt. Een test bestond uit het uitproberen van alle geïmplementeerde functies op mijn eigen toestel, omdat het WeekCal framework wel goed was getest was dit mogelijk.

2.1. Indien de implementatie wel correct was, maar de functie in het WeekCal framework niet moest ik een aanpassing maken in het framework. Deze aanpassingen waren vooral stukken code die net niet werkten zoals dat eigenlijk moest in de Week Calendar app.

Ik kon vervolgens de functie in het framework aanpassen en hierna direct testen, dit is waar test driven-development goed heeft uitgepakt. Na iedere aanpassing kon ik een volledige testrun doen om te zien of het framework nog naar behoren werkte. In een aantal gevallen faalden bepaalde tests door aanpassingen die ik had gemaakt, de testrun had mij hier direct van op de hoogte gebracht waardoor ik de fouten kon oplossen. Als ik niet met TDD had gewerkt had ik deze fouten pas op een later moment opgemerkt, dit had veel extra debugging werk en eventuele herschrijvingen gekost.

3. Oude code verwijderen

Na een implementatie te testen met positief resultaat kon ik de oude code verwijderen die tot dit punt nog bestond of uitgecomment was. De oude code werd nog altijd opgeslagen in GitLab voor het geval ik de oude nog nodig had.

4. Overgebleven deprecaties oplossen
   Zoals beschreven in hoofdstuk 4.2.4 had ik al een analyse gedaan van deprecated functies en onveilige code. Op dit moment kon ik gebruik maken van het deprecation modernisation report als handleiding voor het implementeren van de gemoderniseerde functies. Dit bestond in bijna alle gevallen uit het importeren van het WeekCal framework en de nieuwe functie aanroepen

Aan het einde van het implementeren van de modules had het WeekCal framework volledig geïmplementeerd in de Week Calendar app zonder problemen te veroorzaken in het WeekCal framework.

4.3.1.3 Debugging met XCode instruments

Het was nu tijd voor de laatste tests waar ik het eerder in dit verslag nog niet over heb gehad. XCode heeft een collectie van tools genaamd XCode Instruments waarmee het mogelijk is om applicaties te profilen voor CPU en geheugen gebruik, twee aspecten die een zeer groot effect hebben op mobiele applicaties. Toestellen zoals iPhones en iPads hebben namelijk een gelimiteerde hoeveelheid RAM geheugen beschikbaar voor mobiele applicaties, als een mobiele applicatie over een dergelijk limiet heen gaat zal de applicatie crashen. Om deze reden heb ik met XCode instruments de CPU en het geheugen gebruik van de Week Calendar met framework implementatie gemeten om achter pijnpunten in het WeekCal framework te komen die na de implementatie aanwezig waren.

De resultaten hiervan waren over het algemeen zeer positief, ik heb geen geheugen lekken gevonden maar wel een aantal CPU pijnpunten. Wanneer een gebruiker veel contacten had toegevoegd aan een afspraak kon het meerdere seconden duren voordat deze contacten worden gekoppeld aan de Contacts module, bij 5 contacten was dit merkbaar, bij 10 aanzienlijk en bij 20 of meer extreem.

Afbeelding 70. XCode Instruments CPU Timer. findLinkedPersonsInString functie

In afbeelding 70 wordt een deel van de tool CPU Timer uit XCode instruments afgebeeld. De functie '*findLinkedPersonsInString*' verbruikt in dit geval 1045ms aan CPU tijd, wat 6,2% is van de gehele CPU tijd. Dit is zeer veel en moest worden opgelost, dit heb ik gedaan door de functie te optimaliseren en op een achtergrond thread te laten draaien. Aan het einde van de optimalisatie heb ik nogmaals de CPU tijd van de functie opgezocht via XCode instruments, het resultaat staat afgebeeld in afbeelding 71:

Afbeelding 71. Na optimalisatie

4.3.1.4 WeekCal framework implementatie in iPad Week Calendar app

Na het WeekCal framework volledig te implementeren in de iPhone Week Calendar app had ik nog weinig tijd over om het WeekCal framework ook in de iPad app te implementeren. Het was op dit moment al duidelijk dat ik de implementatie niet zou kunnen voltooien binnen de tijd die ik hier nog aan kon besteden.
Om deze reden heb ik samen met mijn afstudeermentor besproken wat de volgende stap moest worden. Hieruit is de beslissing gekomen om zoveel mogelijk modules te implementeren in de iPad Week Calendar app en het product werkend af te leveren.

Ik heb vervolgens de Date utilities en networking modules geïmplementeerd en getest. De iPad implementatie is op dit moment dus niet voltooid maar wel werkend, er zijn geen problemen en de implementatie kan op ieder moment worden hervat.

4.3.2 Eindproduct

Het eindresultaat is de iPhone Week Calendar app werkend met het geïmplementeerde WeekCal framework. De implementatie van het WeekCal framework in de iPad is nog niet voltooid. De volgende mijlpalen heb ik bereikt:

iPhone Week Calendar app kan worden geüpdatet via het WeekCal framework

Omdat de iPhone Week Calendar nu het framework heeft geïntegreerd kan de iPhone Week Calendar applicatie worden geüpdatet via het WeekCal framework. Wanneer het WeekCal framework is aangepast hoeft het vernieuwde framework slechts te worden gesleept naar de iPhone applicatie om een update to voltooien.

Door tijdgebrek is het niet gelukt om ook de iPad applicatie volledig te integreren met het WeekCal framework, slechts de Date utilities en networking module zijn geïntegreerd. In de toekomst kan de rest van het framework wel worden geïntegreerd in de iPad applicatie.

Het WeekCal framework heeft een eigen git repository, er kan dus door meerdere personen tegelijk aan gewerkt worden. In de oude situatie ontstonden er problemen wanneer er door meerdere ontwikkelaars werken aan de iPhone en iPad applicaties omdat dit twee verschillende repositories waren. Zodra de iPad applicatie volledig is geïntegreerd zal ook dit probleem volledig zijn opgelost, het probleem is nu gedeeltelijk opgelost omdat het wel mogelijk is om nieuwe functionaliteit toe te voegen in het WeekCal framework. De nieuwe functionaliteit is dan direct beschikbaar voor zowel de iPhone als iPad applicatie (na het vernieuwde framework in de iPhone en iPad applicatie te slepen).

Alle deprecated code gemoderniseerd

In de iPhone Week Calendar app is alle verouderde (deprecated) code vervangen met moderne varianten. Dit levert verbeterde stabiliteit en performance op. De exacte moderniseringen zijn te vinden in het deprecation modernisation report.

Minder regels code voor zelfde functionaliteit
Om een goede vergelijking te maken van de hoeveelheid bespaarde code regels is hieronder een tabel met de precieze informatie geplaatst.

Taal	Klassen	Regels Code	Verschil in regels met oude situatie
Objective-C	580	102.066	**-21.662**
C/C++ Header	595	9.885	**-2136**
C++	2	5.346	0
Swift	20	655	0
Objective-C++	2	421	0
Totaal zonder WeekCal framework	1.199	118.373	**-23.798**
WeekCal Framework (Swift)	121	6.239	**+6.239**
Totaal met WeekCal framework	1.320	124.612	**-17.559**

Tabel 17. Regels code na framework implementatie

Zoals in tabel 17 staat beschreven heeft de Week Calendar app na de framework implementatie 23.798 minder regels code. Veel van deze code zat verscholen achter het AddressBook framework dat nu is omgezet in het Contacts framework.

Natuurlijk is er ook code bijgekomen in de vorm van Swift code in het WeekCal framework. Het WeekCal framework beslaat 6.239 regels code. De totale winst aan regels code (dus hoeveel er bespaard zijn) is dus 23.798 - 6.239 = 17.559

Status van het in productie brengen
Zoals eerder beschreven is de iPad week calendar implementatie nog niet voltooid. Dit heeft echter geen effect op de iPhone week calendar implementatie die wel in productie kan worden gebracht. Ik heb niet zelf het product in productie gebracht, dit ligt namelijk buiten de scope van mijn project.

WeekCal is van plan om eerst de iPhone Week Calendar app met het WeekCal framework intern te beta testen. Als de beta test goed verloopt zal de iPhone Week Calendar app met het WeekCal framework in productie worden gebracht. Als er zich geen problemen voordoen (veroorzaakt door het WeekCal framework) zal de iPad WeekCal framework implementatie worden voltooid, beta tested en tot slot in productie worden gebracht.

Zodra deze plannen zijn voltooid is het WeekCal framework en de implementatie in productie, wat mijn project tot een succesvol einde brengt.

5. Evaluatie

In dit hoofdstuk ga ik mijn ontwikkelde producten, proces en beroepstaken evalueren. In het algemeen ben ik tevreden over mijn afstudeer opdracht en ook trots op het eindresultaat.

5.1.Productevaluatie

5.1.1 Eindresultaat

WeekCal was tevreden met het behaalde eindresultaat, ik heb het project grotendeels voltooid. Om een goede evaluatie van het eindresultaat te geven ga ik eerst evalueren of de doelen van het WeekCal framework project zijn bereikt.

1. Het vergemakkelijken van updates van de iPad en iPhone week calendar applicatie

In de nieuwe situatie kan de iPhone applicatie worden geüpdatet door functionaliteit toe te voegen of aan te passen in het WeekCal framework. De iPad applicatie kan echter momenteel alleen nieuwe functionaliteit verkrijgen en niet direct worden veranderd, dit komt doordat de integratie van het WeekCal framework in de iPad app nog niet is voltooid. Het probleem van de oude situatie waar bestanden moesten worden gelinkt is opgelost en het updaten van de iPad en iPhone applicatie gaat een stuk makkelijker. Dit doel is dus grotendeels bereikt en wanneer de iPad applicatie volledig is geïntegreerd is het doel volledig bereikt.

2. Verzekeren dat iPhone en iPad dezelfde functionaliteit hebben

Dit doel is bereikt op een andere scope dan dat ik aanvankelijk had bedacht. Bij de aanvang van de opdracht had ik in mijn hoofd dat alle functionaliteit van de iPhone en iPad applicatie zouden worden verwerkt in het WeekCal framework. Dit was echter meer werk dan dat ik aankon in de relatief korte afstudeer periode, om deze reden heb ik ook het requirements rapport opgesteld met daarin functionaliteit die wel ontwikkeld ging worden.

Het doel is bereikt op de manier dat de iPhone en iPad applicatie dezelfde functionaliteit aangeboden krijgen via het WeekCal framework, maar niet op de manier dat de applicaties in de huidige situatie de exact zelfde functionaliteit hebben. Meer functionaliteit kan in de toekomst wel worden toegevoegd aan het WeekCal framework om dichter bij dit doel te komen.

Hoewel dit doel dus niet is uitgepakt zoals ik in eerste instantie van plan was ben ik toch tevreden met het resultaat, het WeekCal framework verzekert namelijk wel dat alle nieuwe functionaliteit en alle requirements hetzelfde zijn voor iPhone en iPad.

3. Verouderde code moderniseren

Zoals onder andere duidelijk wordt uit het deprecation modernisation rapport is tijdens het project zeer veel code gemoderniseerd. Alle verouderde code uit de Week Calendar app is gemoderniseerd in het WeekCal framework waarmee dit doel is bereikt.

Na het framework te hebben ontwikkeld heb ik het framework geïmplementeerd in de Week Calendar app, ik ben tevreden over de snelheid en de kwaliteit van de implementatie maar heb ook fouten gemaakt. Zoals eerder al beschreven in hoofdstuk 4.3 kwam ik een aantal fouten tegen die ik had kunnen voorkomen als ik meer onderzoek had gedaan naar de conversie en compatibiliteit tussen Objective-C en Swift. Deze kennis heb ik nu, maar een volgende keer als er een soortgelijke situatie

voordoet zal ik zeker eerst onderzoek doen naar compatibiliteit en conversies. Als ik dat bij dit project had gedaan had dit mij meerdere dagen aan werk kunnen besparen.

Een van de drie hoofddoelen is dus volledig bereikt en twee doelen grotendeels. Wel wist ik op tijd dat doel 2 anders zou uitpakken en heb ik het requirements rapport opgesteld om verdere problemen te voorkomen. Dit probleem had mogelijk voorkomen kunnen worden als ik de opdrachtomschrijving nog duidelijker had gemaakt voordat ik begon aan de afstudeeropdracht, een fout die aan mijn kant lag omdat ik WeekCal niet om genoeg informatie heb gevraagd over de opdracht. Ondanks deze fout ben ik tevreden over de oplossing die ik heb gevonden en het uiteindelijke WeekCal framework dat volledig testbaar, unit-tested en gedocumenteerd is.

5.1.2 Tussen producten

Plan van Aanpak
Het plan van aanpak heeft geholpen met het duidelijk maken van de opdracht, het beoogde resultaat, de tussenproducten en de planning. Het plan heeft de opdracht goed afgebakend en duidelijk gemaakt wat mijn afstudeeropdracht in hield. De globale planning in het plan van aanpak is echter niet volledig realistisch en ik ben er ook veel van afgeweken, vooral het implementeren van het framework heeft meer tijd gekost dan initieel gepland in het plan van aanpak. Dit was echter een globale planning en was alleen bedoeld als globale indicatie van de hoeveel tijd dat ieder product zou kosten.

Requirements Rapport
Het requirements rapport is van groot belang geweest voor de ontwikkeling van het WeekCal framework. Het requirements rapport laat goed zien wat er gemaakt moet worden en behoudt de generieke aard van het WeekCal framework, wat van groot belang was. In dat opzicht was het requirements rapport dus een succes. Er zijn echter wel een aantal tekortkomingen, een aantal requirements heb ik niet goed genoeg gedefinieerd waardoor verwarring kan ontstaan wanneer een andere ontwikkelaar de requirements doorneemt. In dit geval was ik de enige die het rapport gebruikte en kwamen er geen problemen van maar in een volgend project zal ik zeker ervoor zorgen dat alle requirements correct zijn gedefinieerd.

System Design Diagram
Het system design diagram is in het WeekCal framework project een groot en complex diagram geworden. Ik ben zeer tevreden over de gebruikte design patterns en het behoud van een generiek ontwerp. Ik heb het ontwerp ook weinig hoeven veranderen tijdens de ontwikkeling van het WeekCal framework, wat betekent dat de kern van het ontwerp van het begin af aan goed was. Het huidige system design diagram is één groot diagram met alle modules erin verwerkt. Hoewel de modules wel zijn gescheiden is het alsnog lastig om het gehele diagram te overzien. Om deze reden zou ik in een volgend project iedere module een eigen diagram geven om zo het overzicht te kunnen behouden, dat nu een beetje is weggevallen.

Deprecation Modernisation rapport
Het deprecation modernisation rapport is een tussenproduct dat volgens de opdracht niet nodig was, ik heb zelf bedacht als een manier om een overzicht te maken van gemoderniseerde code. Het rapport is duidelijk en geeft een goed inzicht over de hoeveelheid gemoderniseerde code maar is ook een handleiding voor ontwikkelaars van de Week Calendar app die het WeekCal framework gaan gebruiken. In een volgend project zou ik alleen de naam van het rapport duidelijker maken.

Master testplan & test rapportages
Door het master testplan op te stellen heb ik ervoor gezorgd dat het duidelijk is wat voor tests ik ging ontwikkelen voor het WeekCal framework. Hierdoor zijn alle tests consistent en duidelijk voor nieuwe ontwikkelaars, ook is het meteen duidelijk welke gebieden het diepst zijn getest door naar de test risico analyse te kijken. Ik heb echter het master testplan wel haastig en niet zeer secuur opgesteld, ik had namelijk haast om te beginnen met het ontwikkelen van het WeekCal framework. Naast het master testplan heb ik ook periodiek test rapportages opgesteld. De test rapportages geven een duidelijk overzicht van de unit tests op het moment van de rapportage, wanneer de rapportages worden vergeleken is ook een stijgende lijn zichtbaar.

Wat ik beter had kunnen doen is de consistentie en afspraken over de test rapportages, ik ben pas laat begonnen met het maken van rapportages waardoor de eerste rapportage al ongeveer 200 unit tests had. Bij een volgend project zou ik dus zeker eerder beginnen met het maken van rapportages.

5.2.Procesevaluatie

Tijdens het WeekCal framework project heb ik gewerkt met SCRUM als project management methode en TDD als systeemontwikkeling methode, deze methodes zijn uitermate geschikt gebleken voor dit project. Ik heb het gehele project lang mij goed gehouden aan de scrum methode zonder hier veel van af te wijken waardoor de planning altijd constant en consistent is gebleven. Ik heb alle SCRUM sprints consistent uitgevoerd en ben ook tevreden over hoe ik met de methode ben omgegaan.

Wel heb ik mij tijdens het project gerealiseerd dat ik eigenlijk helemaal geen projectmanagement methode nodig had, wat SCRUM uniek maakt zijn de team gerelateerde onderdelen waar ik nauwelijks gebruik van heb gemaakt. In plaats daarvan gebruikte ik alleen de sprint structuur, maar daily standups en sprint reviews stelden vaak niet veel voor. Dit brengt mij tot het punt waar ik nog verder in moet groeien: communicatie. Hoewel mijn stagementor slechts een dag in de week op kantoor was heb ik altijd de mogelijkheid gehad om via het chat programma Slack contact op te nemen, maar ik heb dit niet gestructureerd gedaan. Vaak als ik ergens niet uitkwam of wanneer ik een keuze moest maken sloeg ik de communicatie stap over, waar ik dan later spijt van had.

Als ik daily standups en sprint reviews serieuzer had genomen en dagelijks, of op zijn minst meerdere malen in de week, contact had gemaakt met mijn mentor had het project hoogstwaarschijnlijk beter verlopen. Fouten die ik zelf maak, of het nu qua planning of code is, merk ik zelf niet direct op. Dit is waar communicatie een grote invloed had gemaakt op mijn functioneren, ik ben niet frequent op mijn fouten gewezen waardoor ik mezelf niet genoeg heb verbeterd. Als ik beter had gecommuniceerd was dit hoogstwaarschijnlijk anders geweest, bij volgende projecten zal ik communicatie beter aanpakken door afspraken en een communicatie agenda te maken

Naast de SCRUM methode heb ik mijn project opgedeeld in drie fases die alle drie verantwoordelijk waren voor een deel van het project, dit concept heb ik geleend van de RUP methode. Ik heb de fases afzonderlijk van elkaar uitgevoerd zonder afwijkingen, waardoor ik ook producten in de correcte volgorde heb gemaakt. De producten werden niet door elkaar heen gaan ontwikkelen, hoewel in fase 3 ik soms

werk van fase 2 moest aanpassen en in fase 2 werk van fase 1. Dit waren altijd kleine aanpassingen die ik van tevoren al had verwacht; de planning was hierop voorbereid door middel van uitlooptijd.

Zoals eerder beschreven heb ik als systeemontwikkeling methode Test-Driven Development gekozen, wat een van de beste keuzes is geweest van dit project. TDD heeft een goede invloed gemaakt in dit project met als resultaat een volledig testbaar WeekCal framework waar ik trots op ben. Pas in dit project ben ik erachter gekomen dat TDD en testen leuk en belonend zijn, nooit eerder had ik zoveel tijd besteed aan testen. Ik heb me volledig gehouden aan het principe waar eerst de test wordt geschreven en vervolgens pas de implementatie waardoor ik een consistent testbaar product heb verkregen.

De gebruikte project managementmethode en systeemontwikkeling methode heb ik dus goed gevolgd waardoor ik een kwaliteitsproduct heb geleverd waar ik trots op ben. Bij een soortgelijk project zou ik ook zeker weer deze combinatie kiezen met nadruk op het correct volgen van de methodes, als ik namelijk de methodes niet goed had gevolgd was het eindresultaat heel anders geweest.

Qua planning verliep het project tijdens de framework ontwikkeling fase goed, de grootste factor hiervoor waren SCRUM sprints. Door per sprint een module te ontwikkelen kon ik accuraat schatten hoeveel tijd ik over had en hoeveel ik nodig had. De meeste modules waren qua werk namelijk ongeveer gelijk. Wanneer ik bijvoorbeeld nog 3 modules moest ontwikkelen wist ik dat dit 3 sprints zou kosten, waardoor ik de planning goed kon afstemmen.

Ik ben nooit in tijdsnood gekomen, hoewel het wel een aantal keren is voorgekomen dat een bepaalde modules meer tijd kostte dan verwacht. Een voorbeeld hiervan is EventStore module, deze module heeft mij ongeveer 5 dagen langer gekost dan verwacht. Ik had echter altijd een uitlooptijd van een week per module ingepland, de uitlooptijd heb ik nooit overschreden en heeft dus alle uitgelopen modules opgevangen.

Tijdens het implementeren van het WeekCal framework in de iPhone en iPad Week Calendar applicaties is de planning helaas minder goed verlopen. Hierdoor heb ik de iPad app WeekCal framework implementatie niet binnen de gegeven tijd kunnen afmaken, wat zonde was. Tijdens het implementeren van het WeekCal framework in de iPhone app heb ik teveel tijd gestoken in het moderniseren van code, hierdoor heb ik mezelf niet goed aan de planning gehouden.

5.3.Evaluatie beroepstaken

5.3.1 Uitvoeren analyse door definitie van requirements

Deze beroepstaak heb ik met niveau 3 afgerond. Tijdens het WeekCal framework project bestond deze beroepstaak uit definiëren van de requirements en het opstellen van een requirements rapport. Het opstellen van de requirements was lastig omdat het WeekCal framework generiek moest zijn en toch veel functionaliteit nodig had, ik heb dit goed opgelost door veel te letten op de generiekheid van de requirements. Hoewel ik met mijn afstudeer mentor gesproken heb over de requirements heb ik deze wel zelfstandig en zonder leiding opgesteld. De requirements sloten goed aan op de rest van het WeekCal framework en zijn de bouwstenen van het project geworden.

Bij een volgend project zou ik, nog voordat ik begin met de opdracht, al een beknopte opgave maken van de requirements zodat ik mij beter kan voorbereiden. Bij dit project had dat ook de inconsistentie van doel 2 uit hoofdstuk 5.1.1 kunnen voorkomen.

5.3.2 Ontwerpen Systeemdeel

De ‘ontwerpen systeemdeel’ beroepstaak heb ik met niveau 4 afgerond. Deze beroepstaak bestond bij het WeekCal framework project uit het omzetten van het requirements rapport in een System Design Diagram. Hierbij moest ik een generiek ontwerp ontwikkelen waarbij ik rekening moest houden met uitbreidbaarheid, mogelijkheden binnen de XCode omgeving en functionaliteit. Binnen het System Design Diagram heb ik meerdere design patterns gebruikt die goed hebben uitgepakt bij het behalen van de doelen voor het WeekCal framework. Ik heb het system design diagram zelfstandig ontwikkeld en ben tevreden over de uitkomst, een generiek ontwerp met correct toegepaste design patterns.

De keuze voor design patterns heb ik wel vaak snel gemaakt zonder veel onderzoek te doen naar alternatieve design patterns. De gebruikte patterns kwamen in dit geval goed tot hun recht, maar als ik meer onderzoek had gedaan had ik wellicht design patterns kunnen gebruiken die meer geschikt waren voor het WeekCal framework. Bij een volgend project zou ik dan ook tijdens het ontwerpen meer onderzoek doen naar design patterns in plaats van het eerste passende pattern te kiezen.

5.3.3 Bouwen Applicatie

De ‘Bouwen Applicatie’ beroepstaak heb ik met niveau 4 afgerond. Deze beroepstaak bestond bij het WeekCal framework uit het ontwikkelen van het WeekCal framework met de Swift programmeertaal en het implementeren van het WeekCal framework in de Week Calendar app met de programmeertaal Objective-C. Alle code binnen het project heb ik zelfstandig zonder begeleiding geschreven.

In het begin van het ontwikkelen van het WeekCal framework zag ik Swift als een gekke programmeertaal met lastige concepten als optionals en limieten bij inheritance. Naarmate ik meer in Swift ging programmeren begon ik de taal niet meer te zien als een gekke syntax, maar eerder als een nieuwe manier om zo betrouwbaar en veilig mogelijke code te schrijven. De concepten van optionals en het sterk aangemoedigde gebruik van interfaces (composition) begonnen steeds logischer te worden en aan het einde van het project kon ik eigenlijk niet meer zonder. Waar ik eerst begon met forced unwrapping ging ik over op guards, ik kwam erachter dat de naam van een functie in iOS veel belangrijker is dan in bijvoorbeeld Java. Kortom, waar ik eerst alleen probeerde te vechten tegen Swift heb ik nu geleerd om het te omarmen.

Swift was voor mij als java ontwikkelaar een hele transitie, een grote verandering maar wel een heel leuke verandering. Swift heeft mij laten zien dat de syntax van een programmeertaal meer kan zijn dan alleen de manier waarop code geschreven moet worden.

Naast het ontwikkelen van het framework moest ik ook het framework integreren waarbij ik dus moest omschakelen van programmeertaal Swift naar Objective-C. Hierbij heb ik lastige problemen met memory management en bridging moeten oplossen. Toen ik net begon aan de framework implementatie had ik nog geen ervaring in Objective-C, nu, aan het einde van de implementatie, ben ik er goed bekend mee. Omdat Objective-C niet vergevingsgezind, moeilijk te debuggen en uniek qua syntax is werd ik hard op de feiten gedrukt en moest ik snel beter beter worden in Objective-C. Ik heb dit goed opgepakt en heb veel geleerd over het belang van namen in Objective-C, hoe ik om moest gaan met manuaal memory management en ik kan nu goed omgaan met de unieke syntax van Objective-C.

Ik ben trots op de code die ik heb geschreven en ben blij dat ik alle problemen heb opgelost. Tot slot heb ik ook het gehele project lang een nette git repository bijgehouden die ik vaak heb gebruikt om fouten op te lossen.

5.3.4 Uitvoeren en rapporteren over het testproces

Deze beroepstaak heb ik met niveau 3 afgerond. Tijdens het WeekCal framework project heeft deze beroepstaak een grote rol gespeeld. Zoals eerder beschreven heb ik het project uitgevoerd met als systeemontwikkeling methode Test-Driven Development. Dit betekent dat ik constant unit-tests heb geschreven en de beroepstaak Bouwen en Testen gelijk met elkaar liepen. Ik heb nooit eerder in een project zoveel getest als in dit project. Met een hoge code coverage en een volledig testbaar WeekCal framework ben ik zeer tevreden en blij met de uitkomst van deze beroepstaak. Bij het ontwikkelen van de unit tests heb ik gebruik gemaakt van unit-tests testontwerp methodes zoals syntax-test en equivalentieklassen-test, deze technieken kwamen goed overeen met een groot deel van de belangrijke functies. Behalve functionele tests heb ik ook niet functionele unit-tests gemaakt, in dit geval performance-tests. De performance tests gaven een goed inzicht in de performance van het Contacts framework en het lazy factory design pattern.

Naast het uitvoeren van tests heb ik iedere maand (vanaf de 2e maand) een test rapportage opgesteld waarin de unit-tests en code coverage is beschreven. Het uiteindelijke resultaat dat mij in staat heeft gesteld om binnen een minuut een test run uit te voeren heeft veel geholpen bij de implementatie fase en heeft goed gewerkt. Deze beroepstaak heeft mij tijdens dit project geleerd dat testen leuk kan zijn en qua uitvoering van het testproces zou ik een volgende keer dan ook niets veranderen.

Wat ik bij een volgend project anders zou doen is het rapporteren over het testproces. Ik had hier eerder mee kunnen beginnen waardoor de progressie van de unit-tests en de code coverage duidelijker zou zijn geworden. De test rapportages die ik nu heb gemaakt verschillen weinig van elkaar omdat het grootste deel van de unit-tests al geschreven en uitgevoerd waren tegen de tijd dat ik de eerste rapportage had gemaakt.

Referentie lijst

1. Edward Valentini, EVContactsPicker, 2016
   https://github.com/edwardvalentini/EVContactsPicker
2. P. Raba, EPContactsPicker, 2016
   https://github.com/ipraba/EPContactsPicker
3. ICU, ICU documentatie, 2016
   http://userguide.icu-project.org/formatparse/datetime
4. TMap, TMAP Next, 2016
   http://www.tmap.net/tmap-next
5. Apple, Apple Developer Documentation Library, 2016
   https://developer.apple.com/library/ios/navigation/
6. Javin Paul, 5 Reasons to Use Composition over Inheritance in Java and OOP, 2013
   http://javarevisited.blogspot.nl/2013/06/why-favor-composition-over-inheritance-java-oops-design.html
7. Jorge Córdoba, Difference between static class and singleton pattern, 2009
   1. http://stackoverflow.com/a/519530
   2. http://stackoverflow.com/a/519536\
   3. http://stackoverflow.com/a/12367609
   4. http://stackoverflow.com/a/519556
8. Apple, Swift API Guidelines, 2016
   https://swift.org/documentation/api-design-guidelines/#naming
9. Apple, Contacts API Reference, 2016
   1. https://developer.apple.com/reference/contacts
   2. https://developer.apple.com/reference/contacts#1773385
   3. https://developer.apple.com/reference/contacts#1773390
10. Apple, AddressBook API Reference, 2016
    https://developer.apple.com/reference/addressbook/
11. Joyce Echessa, CocoaPods: What is it Good For?, 2014
    https://www.sitepoint.com/cocoapods-good/
12. Apple, Observing External Changes to the Calendar Database
    https://developer.apple.com/library/content/documentation/DataManagement/Conceptual/EventKitProgGuide/ObservingChanges/ObservingChanges.html#//apple_ref/doc/uid/TP40009765-CH4-SW1
13. Matt Thompson, NSNotification & NSNotificationCenter, 2013
    http://nshipster.com/nsnotification-and-nsnotificationcenter/
14. Vincent Guerci, Swift: iOS Deployment Target Command Line Flag, 2014
    http://stackoverflow.com/a/24397402
15. Andrés Ibañez, Are You Storing Sensitive Data in NSUserDefaults?, 2014
    https://www.andyibanez.com/nsuserdefaults-not-for-sensitive-data/
16. Apple, UserDefaults API Reference, 2016
    https://developer.apple.com/reference/foundation/userdefaults
17. Jeremiah Jessel, Should I Use Core Data in My App?, 2015
    http://www.learncoredata.com/should-i-use-core-data/
18. Nate Cook, guard & defer, 2015
    http://nshipster.com/guard-and-defer/
19. Bart Jacobs, Five Signs of Code Smell in Swift, 2016
    https://cocoacasts.com/five-signs-of-code-smell-in-swift/
20. Benedikt Terhechte, Tuples in Swift, Advanced Usage and Best Practices, 2015
    https://appventure.me/2015/07/19/tuples-swift-advanced-usage-best-practices/
21. Matt Vague, Swift typealias to the rescue, 2015
    https://medium.com/swift-programming/swift-typealias-to-the-rescue-b1027fc571e3#.sb7doe11z

22. Apple, Framework Reference, 2016
https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Framework.html
23. Joseph Simm, If and when you should use Test-Drive Development, 2016
https://zeroturnaround.com/rebellabs/if-and-when-you-should-use-test-driven-development/
24. Apple, XCTest, 2016
https://developer.apple.com/reference/xctest
25. Jordon Rose & Chris Lattner, Make unsafe pointer nullability explicit using Optional, 2015
https://github.com/apple/swift-evolution/blob/master/proposals/0055-optional-unsafe-pointers.md
26. Swift, Migrating to Swift 2.3 or Swift 3 from Swift 2.2, 2016
https://swift.org/migration-guide/
27. Vojtech Vrbka, Submitting iOS app using beta version of XCode, 2015
http://stackoverflow.com/a/28458280/2078401
28. Laura Brandenburg, When to Write a use Case, 2016
http://www.bridging-the-gap.com/when-would-you-write-a-use-case/

Bijlage

Inhoudsopgave

1. Plan van Aanpak

Plan van aanpak

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum **:** 09 Mei 2016
Status **: N/A**
Versie **: 0.1**

1. Inleiding

Naar aanleiding van het Week Calendar framework project wordt dit plan van aanpak geschreven. Eerst wordt de uit te voeren opdracht beschreven door middel van een opdrachtomschrijving, probleembeschrijving en doelstelling. Vervolgens wordt het project afgebakend en de planning beschreven, tot slot worden de mijlpaalproducten beschreven.

2. Opdrachtomschrijving Bedrijf/Organisatie

Ontwikkelen van een gemeenschappelijk framework voor de applicatie Week Calender bij WeekCal.

3. Probleembeschrijving

WeekCal heeft op dit moment twee verschillende Week Calendar apps in de App Store, voor zowel iPhone als iPad is er namelijk een aparte iOS applicatie ontwikkeld. Beide apps gebruiken zo veel mogelijk dezelfde codebase, beide met specifieke branches in de user interface elementen. De iPhone applicatie heeft meer features en functionaliteit dan zijn iPad tegenhanger. Ook al gebruiken deze applicaties voor een groot deel dezelfde architectuur, worden tijdens het bouw proces beide projecten als individuele codebases behandeld. Het effect hiervan is dat veranderingen die tijdens het ontwikkelen van één van de apps zijn gemaakt niet automatisch mee veranderen op de andere app.

De huidige oplossing in het delen van de codebase zorgt voor de volgende problemen:

- Gelijktijdige ontwikkeling aan de beide apps is complex (of onmogelijk);
- Het aantal ontwikkelaars wat gelijktijdig aan de apps kan werken wordt beperkt door de snel toenemende complexiteit bij het doorvoeren van aanpassingen in de codebase;
- Nieuwe ontwikkelingen kunnen minder eenvoudig op de achtergrond als beta-feature meegenomen wordt in een normale release;
- Code kan niet hergebruikt worden in andere apps zoals bijvoorbeeld die van Datumprikker.

4. Doelstelling

Het doel van dit project is het ontwikkelen van een WeekCal framework dat alle functionaliteit aanbiedt van de al bestaande iPhone en iPad Week Calendar iOS app, m.u.v. de user interfaces. Vervolgens kunnen de iPhone en iPad applicatie beiden het framework gebruiken en dus, op user interface na, hun codebase weghalen. Wanneer er een update/verandering is binnen het framework kunnen beide applicaties direct worden geüpdatet met de nieuwste codebase, in plaats van dat beide applicaties hun codebase moeten aanpassen. Het doel dat hiermee bereikt wordt is het vergemakkelijken van het onderhoud van de iPhone en iPad

applicatie. Doordat er slechts één gescheiden codebase als framework is, is consistentie geen probleem meer, kunnen meerdere ontwikkelaars tegelijk eraan werken en kan code gemakkelijk worden hergebruikt door andere applicaties die het framework gebruiken. Door deze verandering kost het onderhouden van de Week Calendar applicaties minder tijd en geld.
Tijdens het ontwikkelen van het WeekCal framework is het de bedoeling dat de functionaliteit die in het framework worden geplaatst ook direct naar Swift code worden omgezet en wanneer deze functionaliteit van oude API's gebruik maken, deze moderniseren.

5. Afbakening

Het project houdt de volgende onderdelen in:

- Code base van iPad en iPhone samenvoegen
- Objective-C omzetten naar Swift
- Code moderniseren
- Verouderde API’s omzetten naar moderne API’s
- Testplan & rapport

Het project houdt de volgende onderdelen niet in:

- Cross device UI
- Onderzoek naar Objective-C, en Swift

6. Projectorganisatie

De projectgroep bestaat uit twee personen, namelijk Guus Mulder en Saurabh Garg. De rol-en taak verdeling is als volgt:

- Guus Mulder afstudeerder iOS: Requirements, ontwerpen, bouwen en testen
- Saurabh Garg afstudeer mentor & lead iOS developer: Guus Mulder begeleiden en hulp verlenen bij alle taken van Guus Mulder, indien nodig. Hiernaast controleert Saurabh ook iedere week de status van het project door middel van scrum reviews.

7.Planning

Hieronder is een globale planning beschreven waarin het project dient te worden uitgevoerd.

Week	Opdrachten
1	Plan van aanpak
2	Requirements vaststellen
3 - 4	Design ontwerp modelleren
5 - 10	Bouwen framework, ontwerp bijwerken, unit testing
11-13	Framework volledig testbaar maken
14	Documentatie schrijven
15	Framework implementatie in iPhone app
16-19	Schrijven afstudeerverslag

8. Beschrijving mijlpaalproducten

- Requirements Rapport

Het requirements rapport is een document waarin alle vastgestelde requirements worden beschreven, geprioriteerd in combinatie met use cases die aan de requirements zijn gekoppeld.

- System Design Diagram

Het system design diagram is een UML diagram waarin de structuur en het ontwerp van het WeekCal framework wordt beschreven. Het System Design diagram moet aan alle requirements voldoen en een bouwtekening geven voor het bouwen van het WeekCal framework.

- Testplan & rapportage

Binnen het testplan wordt omschreven op welke wijze het te ontwikkelen project getest zal worden. Er wordt uit test methoden en technieken gekozen om de beste te kiezen voor dit project, ook wordt er beargumenteerd waarom andere technieken niet de beste keuze zijn voor dit project. De test rapportage is een document waarin de resultaten van uitgevoerde tests in worden beschreven.

- WeekCal Framework

Het WeekCal Framework is het grootste onderdeel van dit project. Het is de realisatie van het System Design Diagram en voldoet aan alle requirements uit het requirements rapport.

- Week Calendar app werkend met WeekCal framework

Nadat het WeekCal Framework is ontwikkeld kan het worden gebruikt voor in de Week Calendar App. Dit is echter niet zo gemakkelijk als gewoon het project kopiëren. De Week Calendar app moet volledig worden aangepast om alle functies en methoden uit het framework te gebruiken en moet op een goede en testbare wijze dit uitvoeren.

2. Requirements Rapport

Requirements Rapport

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum **:** 19 Mei 2016
Status **: N/A**
Versie **: 1.0**

1. Inleiding

In dit document worden alle requirements van het WeekCal framework project beschreven. Hierbij wordt per requirement een identificatie nummer en prioriteit gegeven. Er zal worden gewerkt met de MOSCOW techniek voor het prioriteren van de requirements. Tot slot zijn er van belangrijke requirements beknopte use case diagrammen gemaakt.

2. Requirements

Business requirements:

Identificatie nummer	Beschrijving
BR1	Oude API's moeten worden omgezet naar de nieuwe varianten
BR2	Oude (lelijke) code moet worden omgezet naar nette moderne code
BR3	Het framework dient generiek te zijn
BR4	Het framework dient uitbreidbaar te zijn

Requirements

Identificatie nummer	Beschrijving	Functioneel of niet functioneel	Prioriteit
R1	Het framework moet datums kunnen opmaken.	F	M
R2	Het framework moet bij het opmaken van datums meerdere calendars kunnen ondersteunen (persisch, moslim, Hindi)	F	M
R3	Het framework moet datum formaten kunnen lezen uit property lists	F	M
R4	Het framework moet zowel GMT, display en EKEventStore tijdzones ondersteunen	F	M
R5	Het framework moet tussen tijdzones kunnen rekenen	F	M
R6	Het framework moet zowel ICU als Cocoa weekdagen ondersteunen	F	M
R7	Het framework moet zowel Europese als Amerikaanse manier van datums ondersteunen	F	M

Identificatie nummer	Beschrijving	Functioneel of niet functioneel	Prioriteit
R8	Het framework moet zowel de 12 uurs als 24 uurs klok ondersteunen	F	M
R9	Het framework moet met datums kunnen rekenen (dagen, maanden, jaren toevoegen/verwijderen) Verschillen meten. Jaren, Maanden, dagen, weeknummers vinden	F	M
R10	Het framework moet kunnen detecteren of een datum binnen een weekend valt	F	M
R11	Het framework moet tijden kunnen omzetten naar alarmen (push notificaties)	F	M
R12	Het framework moet kunnen loggen daar de console in Debug en Release Mode	F	S
R13	Het framework moet gebruikers preferences managen	F	S
R14	Het framework moet kleuren schema's kunnen lezen uit property lists	F	M
R15	Het framework moet met kleuren kunnen rekenen. (r,g,b aanpassen, luminance, saturatie, grayscale)	F	M
R16	Het framework moet kunnen detecteren of een kleur licht of donker is	F	S
R17	Het framework moet calendars kunnen ophalen	F	M
R18	Het framework moet calendars zichtbaar en onzichtbaar kunnen maken	F	M

Identificatie nummer	Beschrijving	Functioneel of niet functioneel	Prioriteit
R19	Het framework moet EKEvents kunnen toevoegen/wijzigen/ verwijderen aan calendars	F	M
R20	Het framework moet kunnen detecteren welke opties een calendar heeft, en of deze meer heeft dan een andere calendar.	F	S
R21	Het framework moet efficiënt integers naar string kunnen omzetten	F	S
R22	Het framework moet kunnen detecteren of een string Right-to-left is (arabisch)	F	M
R23	Het framework moet strings compatible kunnen met voor HTTP requests	F	M
R24	Het framework moet kunnen detecteren of een string HTML is	F	C
R25	Het framework moet strings kunnen splitten	F	S
R26	Het framework moet strings kunnen renderen naar CoreGraphics context	F	M
R27	Het framework moet strings kunnen omzetten van en naar base64	F	C
R28	Het framework moet XML in strings kunnen supporten	F	S
R29	Het framework moet Emoji kunnen detecteren in Strings	F	W
R30	Het framework moet NSData van en naar base64 kunnen omzetten	F	C
R31	Het framework moet indexsets, arrays en dictionaries kunnen aanmaken, filteren en muteren	F	M

Identificatie nummer	Beschrijving	Functioneel of niet functioneel	Prioriteit
R32	Het framework moet de huidige locatie van de gebruiker kunnen bepalen	F	C
R33	Het framework moet teksten kunnen vertalen (localization)	F	M
R34	Het framework moet property lists kunnen inlezen en schrijven	F	M
R35	Het framework moet rauwe bestanden kunnen inlezen en schrijven	F	M
R36	Het framework moet HTTP posts en gets kunnen sturen en reageren op hun reacties	F	C
R37	Het framework moet contacten, groepen en containers uit de contacts app kunnen ophalen	F	M
R38	Het framework moet thumbnail images kunnen maken voor contacten zoals iOS 9 dat doet in de contacts app	F	M
R39	Het framework moet contacten kunnen zoeken op naam, email, telefoonnummer of closure	F	M
R40	Het framework moet de Contact Picker kunnen opstarten en erop reageren	F	M
R41	Het framework moet de contact controller kunnen opstarten en erop reageren	F	M
R42	Het framework moet de verjaardagen van contacten kunnen detecteren; en deze omzetten in EKEvents en notificaties	F	S

Identificatie nummer	Beschrijving	Functioneel of niet functioneel	Prioriteit
R43	Het framework moet in-app purchases kunnen opvragen van iTunes	F	M
R44	Het framework moet in-app purchase aankopen kunnen starten	F	M
R45	Het framework moet aankopen herstellen kunnen starten	F	M
R46	Het framework moet contacten binnen 1 seconde kunnen ophalen	NF	M
R47	Alle belangrijke functies moeten ge-unittest zijn	F	M

Legenda

- F: Functioneel
- NF: Niet Functioneel
- M: Must have
- S: Should Have
- C: Could Have
- W: Want to Have

3. Use Cases

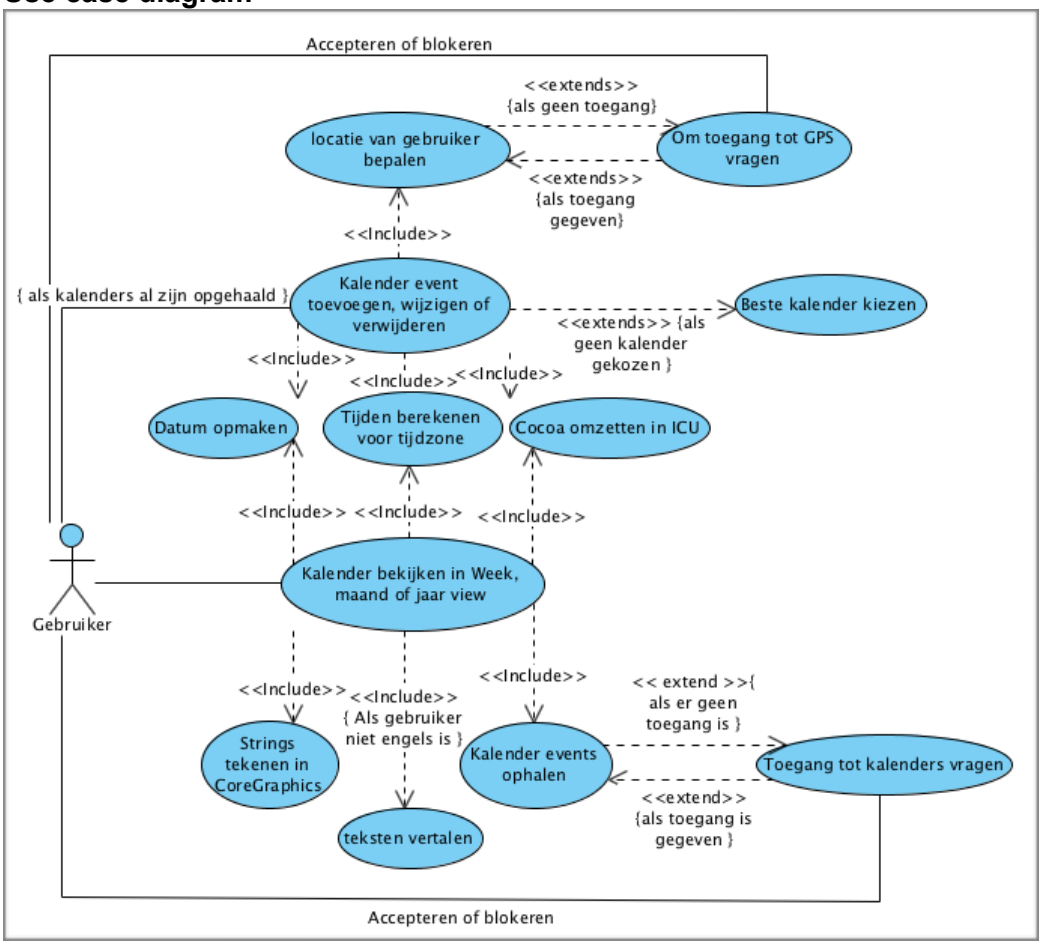
Om een beter beeld te krijgen van de interactie tussen de gebruiker en het WeekCal framework heb ik een aantal use case diagrammen opgesteld . Iedere use case houdt een aantal requirements in die samen een volledig werkend stuk functionaliteit bieden.

Kalenders, tijden en datums.

Requirements

R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R13, R17, R18, R19, R20, R22, R26, R32

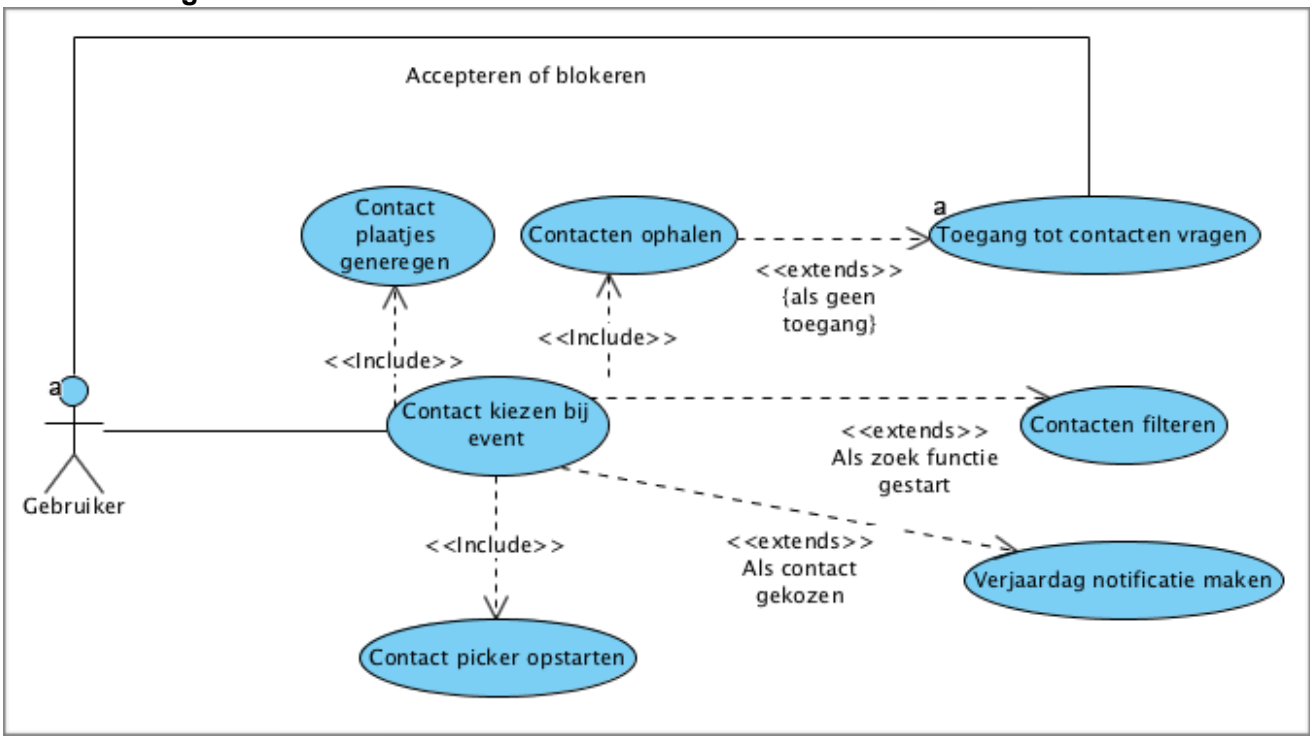
Use case diagram



Contacten

Requirements
R37, R38, R39, R40, R41, R42

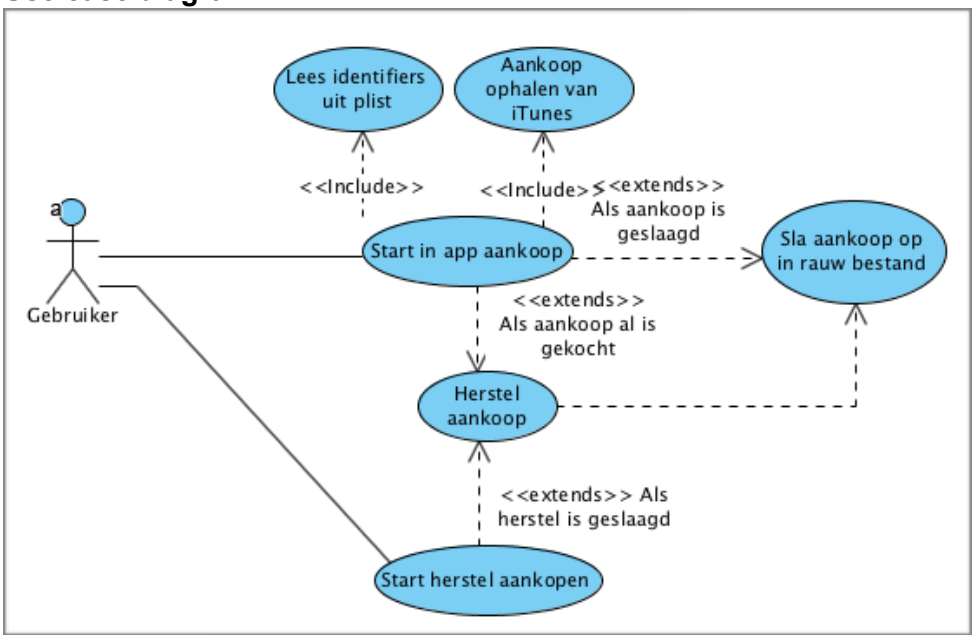
Use case diagram



In app purchases

Requirements
R34, R35, R43, R44, R45

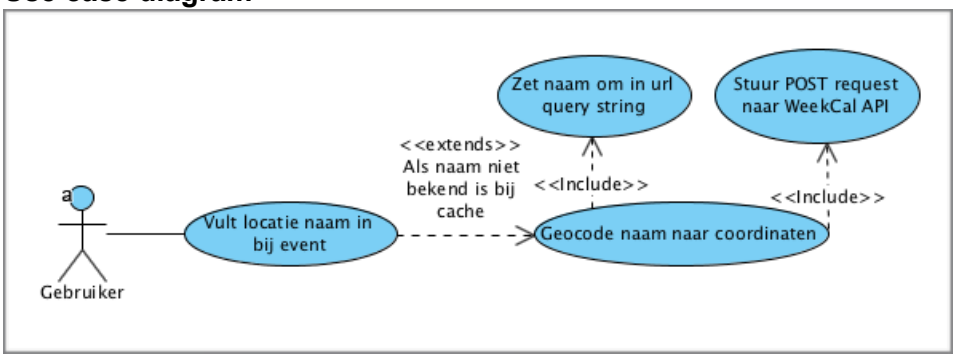
Use case diagram



Netwerk

Requirements
R23, R36

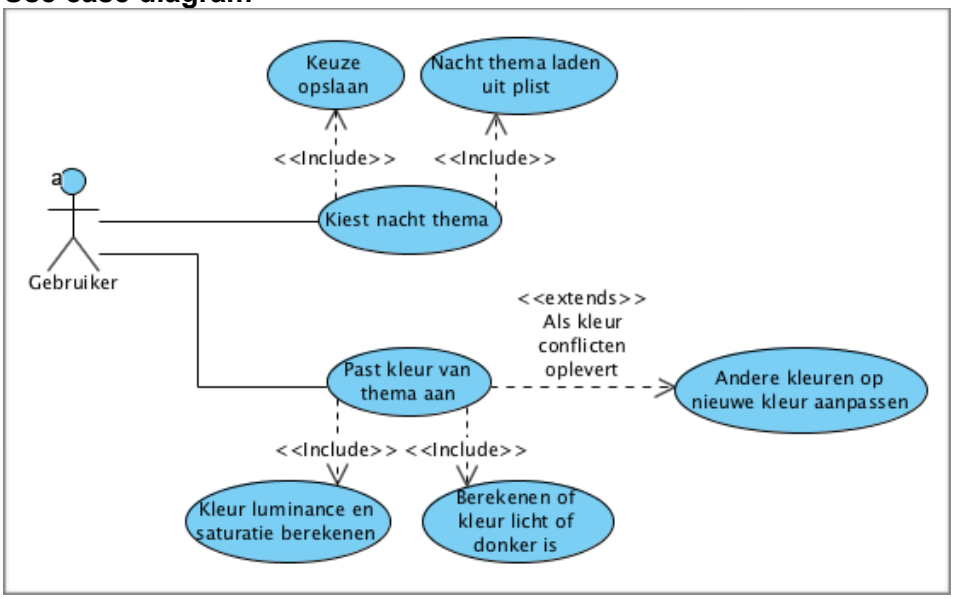
Use case diagram



Theming

Requirements
R13, R14, R15, R16

Use case diagram



3. Master Test Plan

Master test plan

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum : 29 Mei 2016
Status : **N/A**
Versie : **1.0**

1. Introductie

1.1 Project Doel

Het doel van dit project is het ontwikkelen van een WeekCal framework dat alle functionaliteit aanbiedt van de al bestaande iPhone en iPad Week Calendar iOS app, m.u.v. de user interfaces. Vervolgens kunnen de iPhone en iPad applicatie beiden het framework gebruiken en dus, op user interface na, hun codebase weghalen. Wanneer er een update/verandering is binnen het framework kunnen beide applicaties direct worden geüpdatet met de nieuwste codebase, in plaats van dat beide applicaties hun codebase moeten aanpassen. Het doel dat hiermee bereikt wordt is het vergemakkelijken van het onderhoud van de iPhone en iPad applicatie. Doordat er slechts één gescheiden codebase als framework is, is consistentie geen probleem meer, kunnen meerdere ontwikkelaars tegelijk eraan werken en kan code gemakkelijk worden hergebruikt door andere applicaties die het framework gebruiken. Door deze verandering kost het onderhouden van de Week Calendar applicaties minder tijd en geld.

Tijdens het ontwikkelen van het WeekCal framework is het de bedoeling dat de functionaliteit die in het framework worden geplaatst ook direct naar Swift code worden omgezet en wanneer deze functionaliteit van oude API's gebruik maken, deze moderniseren.

1.2 Doel van het master test plan

Het doel van het master test plan is het informeren van betrokkenen van het test proces over de aanpak, activiteiten en producten die worden opgeleverd voor het project. Het master test plan beschrijft welke methode er zal worden gebruikt om te testen, hoe er wordt bepaald hoe diep er getest wordt en op welke wijze dit wordt uitgevoerd.

2. Opdracht formulering

2.1 Opdracht

Ontwikkelen van een gemeenschappelijk framework voor de applicatie Week Calender bij WeekCal.

2.2 Scope

2.2.1 Binnen scope

- Managen van project
- Ontwerpen framework
- Ontwikkelen framework
- Testen framework

2.2.2 Buiten scope

- Reorganisaties
- Activiteiten die door anderen worden uitgevoerd
- Omgeving veranderingen buiten het project

2.3 Acceptanten en acceptatie criteria

2.3.1 Acceptanten

In de onderstaande tabel worden de acceptanten van het WeekCal Framework weergegeven:

Naam	Functie
Guus Mulder	Afstudeerder
Saurabh Garg	Afstudeer Mentor

2.3.2 Acceptatie criteria

In de onderstaande tabel wordt weergegeven welke acceptatie criteria er voor het WeekCal Framework zijn.

Descriptie
Alle unit tests succeeden
Alle functionaliteit met risico’s is unit tested

3. Documentatie

In dit hoofdstuk wordt de documentatie die met het master test plan te maken heeft beschreven.

3.1 Basis voor het master test plan

De volgende documenten worden gebruikt als bron voor het master test plan

Document naam	Versie	Datum	Auteur
Plan van aanpak	1	9 mei	Guus Mulder
Requirements Rapport	1	15 mei	Guus Mulder

3.2 Standaarden

De volgende standaarden zullen worden gebruikt binnen dit test plan.

Document naam	Versie	Datum	Auteur
TMap® Next for result driven testing	1e editie	2006	T. Koomen, L. van der Aalst, B. Broekman en M. Vroon

4. Test strategie

De hoeveelheid tijd om om te testen is gelimiteerd; niet alles kan op eenzelfde diepte worden getest. Dit betekend dat er keuzes gemaakt moeten worden voor wat er dieper getest wordt en wat minder diep.

De test strategie is gebaseerd op risico's. Dit betekend dat hoe hoger het risico van een bepaalde module / functionaliteit is hoe dieper deze getest wordt. Hoe lager het risico, hoe minder er wordt getest. Wanneer er geen risico is wordt er ook niet getest.

Om deze risico's te achterhalen wordt er in 4.1 een product risico analyse analyse uitgevoerd.

4.1 Product risico analyse

De product risico's zijn opgesteld in collaboratie met de stage mentor Saurabh Garg.

Deze product risico analyse (PRA) bestaat uit twee stappen:

- Inventariseren van relevante risico's
- Risico's classificeren

De volledige test risico analyse is te vinden in de bijlage.

Tijdens het opstelling van de risico's zijn ook de volgende testdoelen geformuleerd:

Test doel	**Attributen**
Volledige en kwalitatief hoge functionaliteit	Functionaliteit
Functies moeten nooit slechte performance hebben	Performance
Veiligheid van framework moet van hoge kwaliteit zijn	Veiligheid

Risico Tabel

RK staat voor risico klasse, A = hoog, B = middel, C = laag.

De RK wordt bepaald door de schade van de risico's voor de WeekCal app. Wanneer een risico weinig schade met zich meebrengt, zal de risico klasse lager zijn dan wanneer de applicatie bijvoorbeeld niet meer functioneert door een risico.

Attribuut	**RK**	**Argumentatie**
Functionaliteit	A	Volledige en werkende functionaliteit is een van de doelen van dit project. Als deze functionaliteit niet goed is getest doet dit af van het project
Performance	B	Hoewel performance belangrijk is, is dit binnen het project niet een groot probleem. Tenzij er grote fouten worden gemaakt zal performance geen probleem zijn
Veiligheid	A	De veiligheid van gebruikers van het WeekCal framework is van groot belang. Om deze reden moet er goed getest worden op veiligheid

4.2 Test strategie

Voor ieder risico uit de product risico analyse wordt de testklasse beschreven en de diepte van de uit te voeren unit tests.

Test	Klasse	Unit tests
Functionaliteit		
- Datum en tijd calculaties	A	●●●
- Vertalingen	A	●●●
- In-App aankopen	A	●●●
- Calendars	B	●●
- Contacten	B	●●
- Data structuren	A	●●●
- Thema kleuren	C	●
Performance		
- 2D Rendering	B	●●
Veiligheid		
- In-App aankopen	A	●●●
- HTTP & logins	A	●●●

Legenda:

Klasse	Risico klasse (A=hoog risico, B=gemiddeld risico, C=laag risico)
Unit tests	Unit tests
●	Weinig diepte in unit test
●●	Gemiddelde diepte in unit test
●●●	Hoge diepte in unit test

5. Aanpak

In dit hoofdstuk wordt voor iedere test methode beschreven hoe deze wordt aangepakt.

5.1 Test methodes

Voor dit MTP worden de volgende test methodes gebruikt:

Test methode	Goal
Unit tests	Kwaliteit van code hoog houden

5.2 Unit Tests

5.2.1 Doel

Het doel van unit tests is de kwaliteit van de relevante code hoog houden.

5.2.2 Beschrijving

De unit tests worden gebruikt om de kwaliteit van code hoog te houden. Dit is vooral in dit project belangrijk, waar veel code wordt herschreven door een ontwikkelaar die niet de oude code heeft geschreven. Fouten kunnen snel worden gemaakt en unit tests zorgen ervoor dat deze worden opgemerkt en voorkomen. Hoe dieper de testdiepte hoe meer paden per functie er worden getest. Hoe lager de testdiepte, hoe minder.

5.2.3 Test omgeving

Unit tests dienen op de iOS emulator van Xcode te worden uitgevoerd met iOS 9.3

De unit tests kunnen ook worden uitgevoerd op een fysieke telefoon, maar dit zal leiden tot tests die falen doordat talen anders zullen zijn.

5.3 Test producten

De op te leveren producten zijn:

Fase	Product
Planning	Master Test plan
Management	Test Rapportage maandelijks
Executie	Wanneer succes, de unit test. Wanneer gefaald, unit test verbeteren tot hij succeed

6. Infrastructuur

6.1 Test Omgevingen

Test level	Omgeving	Benodigdheden
Unit Tests	iPhone	iPhone6 of hoger
Unit tests	Emulator	Xcode 7

6.2 Test Tools

Test level	Test tool
Unit Tests	XCode 7, XCTest framework

7. Management

7.1 Test proces management

Tijdens het ontwikkelen van het framework project wordt eerst een unit test aangemaakt en vervolgens pas de bijbehorende functionaliteit. Op die manier is het zeker dat de gewenste uitkomst gelijk is aan wat de functie geeft.

7.2 Test infrastructuur management

Er is geen grote infrastructuur nodig. Slechts een computer met OSX 10.11 waarop Xcode is geïnstalleerd.

7.3 Test product management

Unit tests zullen worden bewaard binnen het framework project. Het framework wordt bijgehouden via git is is daardoor beschermd van verliezen.

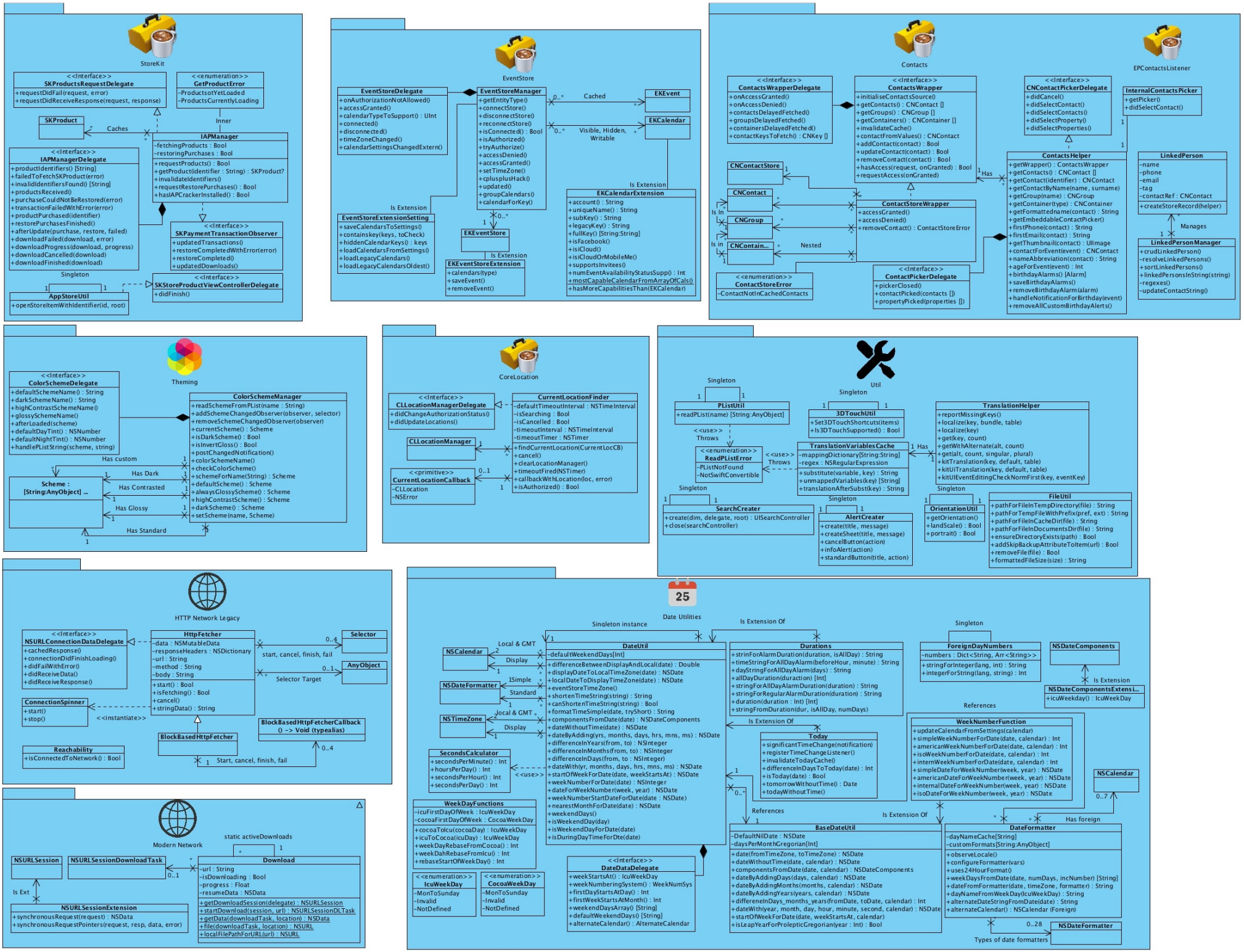
7.4 Defecten procedure

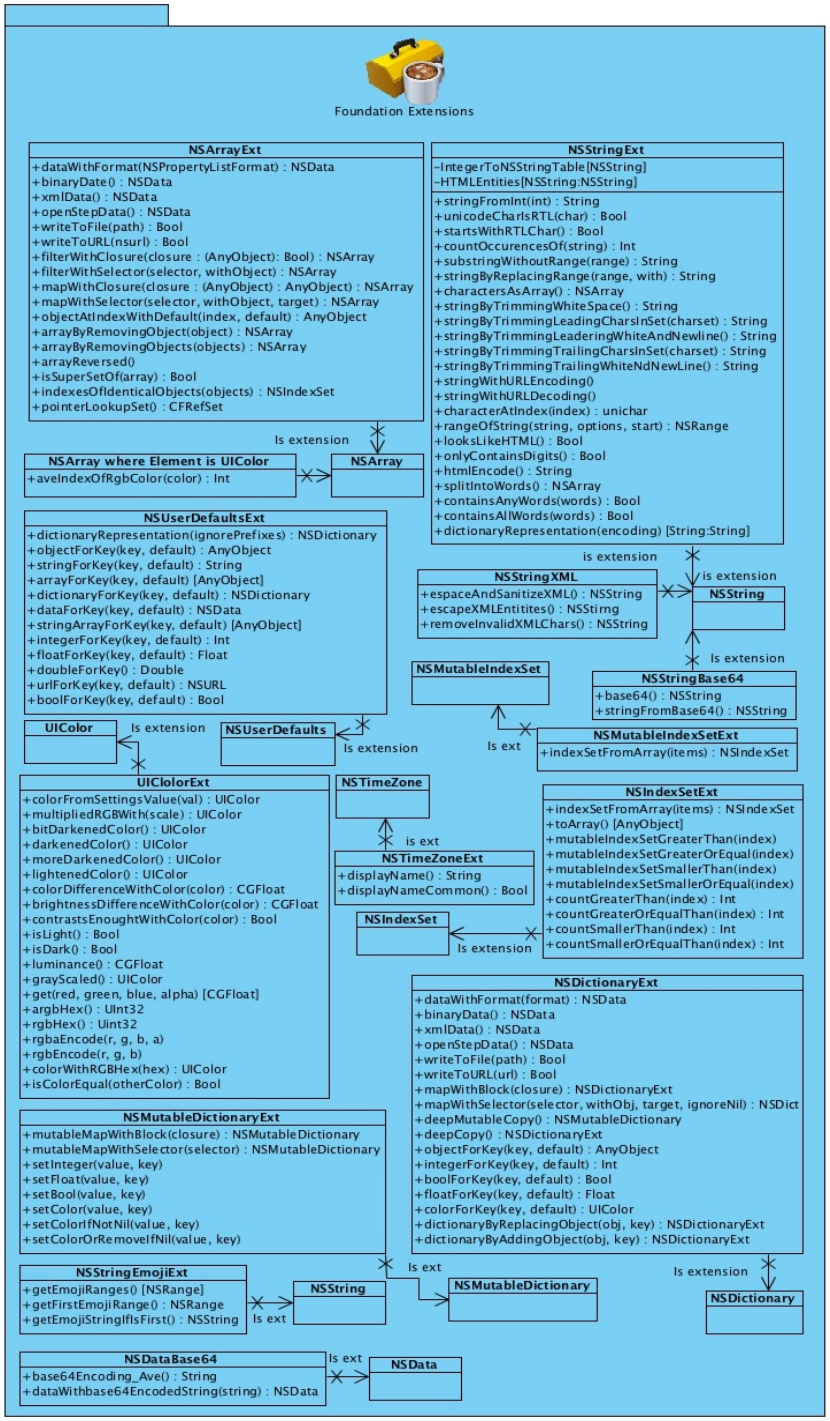
Bij defecten worden tests en code aangepast totdat de tests niet meer defect zijn.

Testrisico Analyse

Module/Mehode/functie	Risico Niveau	Impact	Frequentie
Datum & Tijd calculaties	Kritiek	Hoog	Hoog
Vertalingen	Hoog	Middel	Hoog
In-app aankopen	Kritiek	Hoog	Laag
Calendars	Hoog	Hoog	Hoog
Contacten	Middel	Middel	Middel
Data Structuren	Kritiek	Hoog	Hoog
Thema kleuren	Laag	Laag	Hoog
2D Rendering	Kritiek	Hoog	Hoog
HTTP & Logins	Middel	Middel	Laag

4. System Design Diagram




Foundation Extensions
NSArrayExt
+dataWithFormat(NSPropertyListFormat) : NSData
+binaryDate() : NSData
+xmlData() : NSData
+openStepData() : NSData
+writeToFile(path) : Bool
+writeToURL(nsurl) : Bool
+filterWithClosure(closure : (AnyObject): Bool) : NSArray
+filterWithSelector(selector, withObject) : NSArray
+mapWithClosure(closure : (AnyObject) : AnyObject) : NSArray
+mapWithSelector(selector, withObject, target) : NSArray
+objectAtIndexWithDefault(index, default) : AnyObject
+arrayByRemovingObject(object) : NSArray
+arrayByRemovingObjects(objects) : NSArray
+arrayReversed()
+isSuperSetOf(array) : Bool
+indexesOfIdenticalObjects(objects) : NSIndexSet
+pointerLookupSet() : CFRefSet
NSStringExt
-IntegerToNSStringTable[NSString]
-HTMLEntities[NSString:NSString]
+stringFromInt(int) : String
+unicodeCharIsRTL(char) : Bool
+startsWithRTLChar() : Bool
+countOccurencesOf(string) : Int
+substringWithoutRange(range) : String
+stringByReplacingRange(range, with) : String
+charactersAsArray() : NSArray
+stringByTrimmingWhiteSpace() : String
+stringByTrimmingLeadingCharsInSet(charset) : String
+stringByTrimmingLeaderingWhiteAndNewline() : String
+stringByTrimmingTrailingCharsInSet(charset) : String
+stringByTrimmingTrailingWhiteNdNewLine() : String
+stringWithURLEncoding()
+stringWithURLDecoding()
+characterAtIndex(index) : unichar
+rangeOfString(string, options, start) : NSRange
+looksLikeHTML() : Bool
+onlyContainsDigits() : Bool
+htmlEncode() : String
+splitIntoWords() : NSArray
+containsAnyWords(words) : Bool
+containsAllWords(words) : Bool
+dictionaryRepresentation(encoding) [String:String]
Is extension
NSArray where Element is UIColor
+aveIndexOfRgbColor(color) : Int
NSArray
NSUserDefaultsExt
+dictionaryRepresentation(ignorePrefixes) : NSDictionary
+objectForKey(key, default) : AnyObject
+stringForKey(key, default) : String
+arrayForKey(key, default) [AnyObject]
+dictionaryForKey(key, default) : NSDictionary
+dataForKey(key, default) : NSData
+stringArrayForKey(key, default) [AnyObject]
+integerForKey(key, default) : Int
+floatForKey(key, default) : Float
+doubleForKey() : Double
+urlForKey(key, default) : NSURL
+boolForKey(key, default) : Bool
is extension
NSStringXML
+espaceAndSanitizeXML() : NSString
+escapeXMLEntitites() : NSStirng
+removeInvalidXMLChars() : NSString
is extension
NSString
Is extension
NSStringBase64
+base64() : NSString
+stringFromBase64() : NSString
NSMutableIndexSet
UIColor
Is extension
NSUserDefaults
Is extension
Is ext
NSMutableIndexSetExt
+indexSetFromArray(items) : NSIndexSet
UIClolorExt
+colorFromSettingsValue(val) : UIColor
+multipliedRGBWith(scale) : UIColor
+bitDarkenedColor() : UIColor
+darkenedColor() : UIColor
+moreDarkenedColor() : UIColor
+lightenedColor() : UIColor
+colorDifferenceWithColor(color) : CGFloat
+brightnessDifferenceWithColor(color) : CGFloat
+contrastsEnoughtWithColor(color) : Bool
+isLight() : Bool
+isDark() : Bool
+luminance() : CGFloat
+grayScaled() : UIColor
+get(red, green, blue, alpha) [CGFloat]
+argbHex() : UInt32
+rgbHex() : Uint32
+rgbaEncode(r, g, b, a)
+rgbEncode(r, g, b)
+colorWithRGBHex(hex) : UIColor
+isColorEqual(otherColor) : Bool
NSTimeZone
is ext
NSTimeZoneExt
+displayName() : String
+displayNameCommon() : Bool
NSIndexSetExt
+indexSetFromArray(items) : NSIndexSet
+toArray() [AnyObject]
+mutableIndexSetGreaterThan(index)
+mutableIndexSetGreaterOrEqual(index)
+mutableIndexSetSmallerThan(index)
+mutableIndexSetSmallerOrEqual(index)
+countGreaterThan(index) : Int
+countGreaterOrEqualThan(index) : Int
+countSmallerThan(index) : Int
+countSmallerOrEqualThan(index) : Int
NSIndexSet
Is extension
NSDictionaryExt
+dataWithFormat(format) : NSData
+binaryData() : NSData
+xmlData() : NSData
+openStepData() : NSData
+writeToFile(path) : Bool
+writeToURL(url) : Bool
+mapWithBlock(closure) : NSDictionaryExt
+mapWithSelector(selector, withObj, target, ignoreNil) : NSDict
+deepMutableCopy() : NSMutableDictionary
+deepCopy() : NSDictionaryExt
+objectForKey(key, default) : AnyObject
+integerForKey(key, default) : Int
+boolForKey(key, default) : Bool
+floatForKey(key, default) : Float
+colorForKey(key, default) : UIColor
+dictionaryByReplacingObject(obj, key) : NSDictionaryExt
+dictionaryByAddingObject(obj, key) : NSDictionaryExt
NSMutableDictionaryExt
+mutableMapWithBlock(closure) : NSMutableDictionary
+mutableMapWithSelector(selector) : NSMutableDictionary
+setInteger(value, key)
+setFloat(value, key)
+setBool(value, key)
+setColor(value, key)
+setColorIfNotNil(value, key)
+setColorOrRemoveIfNil(value, key)
Is ext
NSMutableDictionary
Is extension
NSDictionary
NSStringEmojiExt
+getEmojiRanges() [NSRange]
+getFirstEmojiRange() : NSRange
+getEmojiStringIfIsFirst() : NSString
Is ext
NSString
NSDataBase64
+base64Encoding_Ave() : String
+dataWithbase64EncodedString(string) : NSData
Is ext
NSData

5. Rapport bezoek examinator

Rapport bezoek examinator

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum **:** 14 Juni 2016
Status **: N/A**
Versie **: 0.1**

1. **Inleiding**

Naar aanleiding van het bezoek van Arie Toet op 14 Juni 2016 bij WeekCal wordt dit rapport geschreven. Het doel van dit rapport is het vastleggen van gemaakte keuzes en afspraken en beschrijven van het bezoek.

2. Beschrijving bezoek

Op 14 Juni 2016 is Arie Toet om 11:00 op bezoek gekomen bij WeekCal te Wateringen. Tijdens het bezoek heb ik samen mijn afstudeer mentor Saurabh Garg een gesprek gehad met Arie Toet. Het gesprek bestond uit de volgende onderdelen:

- Beschrijving bedrijfs cultuur en gebruikte methodes door Saurabh Garg
- Beschrijving van aanpak en vordering van afstudeeropdracht door Guus Mulder
- Beschrijving van afstudeerdossier en taken / activiteiten die nog gedaan moeten worden tot het einde van het afstuderen

3. Gemaakte keuzes / afspraken

- Elektronisch afstudeerdossier maken, dit had ik nog niet gedaan op het moment van het bezoek
- Bij 45% van de afstudeer periode, voorgangsverslag maken
- Besloten om contact verder via e-mail te laten verlopen
- Volledige codebase hoeft (hoogstwaarschijnlijk) niet te worden gepubliceerd. Kleine stukjes code (snippets) kunnen wel worden gepubliceerd.

6. Voortgangsverslag

Voortgangsverslag

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum **:** 29 Juni 2016
Status **: N/A**
Versie **: 1.0**

1. Inleiding

In dit verslag wordt de voortgang van het WeekCal Framework project beschreven. Er wordt eerst beschreven hoe de opstart van het project is verlopen, vervolgens wordt er verteld hoe het project er op dit moment bij staat. Hierna worden problemen en beslispunten naar boven gehaald om een goed beeld te geven van het pad dat ik heb gevolgd tijdens het project. Tot slot wordt er in de conclusie beschreven hoe ik over de voortgang van project denk en maak ik een schatting over de haalbaarheid van het project in de huidige omgeving.
Het doel van dit document is het informeren van de afstudeer examinatoren over de huidige voortgang, maar ook om mijzelf op de feiten te drukken.

2. Voortgang

De ontwikkeling van het project verloopt in 3 fases; namelijk:
- Opstart
- Ontwikkeling Framework
- Integratie in iPhone applicatie

De opstart fase is op dit moment voltooid en de ontwikkeling framework fase is al ver gevorderd. De integratie in de iPhone applicatie is nog niet begonnen, maar er is wel op voorbereidt. Op welke wijze dit is gedaan wordt verder beschreven in 2.2.

2.1. Opstart

Tijdens de opstart fase heb ik eerst met mijn afstudeer mentor het project nog verder besproken om op die manier een goede projectmanagement methode te kiezen. WeekCal zelf gebruikt op dit moment geen strenge projectmanagement methodes en ik hoefde mij dus ook niet te houden aan een voor mij gekozen methode. Ik heb gekozen om met scrum te werken om de volgende redenen:
- 1 man team: Project wordt door mij alleen uitgevoerd, iedereen is dus direct alles
- Wekelijkse review (als spring review)
- Al ervaring met scrum

Na te hebben gekozen voor scrum moest ik kiezen voor een systeemontwikkeling methode. Ik heb een kort onderzoek gedaan naar beschikbare methodes en vervolgens de meest passende methode gekozen, namelijk: Test driven development. De redenen dat ik voor deze methode heb gekozen zijn:
- Niet bekend met codebase
  Dit betekent dat code snel fout kan worden geïnterpreteerd wat leidt tot fouten. Doordat bij TDD eerst de test wordt geschreven en vervolgens de code zullen deze fouten worden opgevangen. (door middel van een gefaalde unit-test)
- Hoge kwaliteit code
  Het WeekCal framework zal wanneer het in productie gaat direct miljoenen gebruikers hebben. Dit betekent dat fouten in het project snel naar voren zullen komen en problemen kunnen veroorzaken. Hoge kwaliteit code door unit testing voorkomt veel van deze problemen.
- Past goed bij scrum
  Iedereen doet alles, eerst unit tests maken en daarna code past er dus perfect bij.
- Diepe integratie van unit testing in Xcode
  Xcode heeft een geïntegreerd unit testing framework genaamd XCTest. XCTest maakt het gemakkelijk om unit tests bij te houden en kan zelfs code coverage bijhouden.

De omgeving waarin ik het project uit voer was al van te voren bekend en hier heb ik verder niets voor hoeven doen.
Na het bepalen van de te gebruiken project management en ontwikkeling methode heb ik een plan van aanpak geschreven. Hierin staat beschreven wat het doel van het project is, wat de aanpak is en de globale planning.

Tot slot van de opstart fase heb ik de requirements van het project opgesteld. De requirements zijn in samenspraak met mijn afstudeer mentor opgesteld, verdere input was niet nodig omdat mijn afstudeer mentor de lead iOS developer van WeekCal is.

2.2. Ontwikkeling framework

De ontwikkeling framework fase bestaat uit het ontwerpen, bouwen en testen van het WeekCal framework. Hoewel dit niet het integreren van het framework in de iPhone app inhoudt, moet tijdens het ontwerpen van het framework wel rekening gehouden worden met eventuele standaarden of methodes die moeten worden aangehouden om de integratie mogelijk te houden.

De ontwikkelingsfase is begonnen door een system design diagram te maken met als bron de opgestelde requirements. Nadat een basis ontwerp was voltooid heb ik het framework project aangemaakt in Xcode en een scrum backlog aangemaakt. Vanaf dat moment ben ik iteratief via de scrum backlog requirements af gaan lopen. Een iteratie was bijvoorbeeld: Contact Store (adres boek) integratie ontwikkelen. Dit bestond dan uit het maken van unit tests en vervolgens de functionaliteit uit de requirements bouwen. Hierna moest er in de al bestaande codebase worden gekeken naar functionaliteit die ook geporteerd moest worden om het framework aan te vullen. Er werd dan alleen gekeken naar voor die requirement relevante functionaliteit. Echter, bestond dit vaak uit veel functionaliteit en dus veel ObjC-Swift conversie werk; en hiernaast moest deze functionaliteit ook getest worden. Wanneer het bouwen van de requirement was voltooid werd het ontwerp aangepast indien grote verandering moesten plaatsvinden (door de functionaliteit in de oude codebase). Tot slot werd de requirement afgevinkt van de backlog.
Aan het einde van iedere backlog, wat vaak eenmaal per week is, heb ik samen met mijn afstudeer mentor de functionaliteit besproken om ervoor te zorgen dat er niets mist en er geen interpretatiefouten zijn geweest.

Ik werk nog steeds op deze manier om alle requirements af te werken. Op dit moment is circa 70% van de requirements voltooid zonder dat het ontwerp radicale aanpassingen nodig had.

2.3. Beslissingen & Problemen

Hieronder staat een opsomming van beslissingen en design-beslissingen die ik tijdens het project heb gemaakt:

- Requirements diepgang
  Ik heb in samenspraak met mijn afstudeer mentor ervoor gekozen om alleen functionaliteit te bouwen in het framework (zoals In-app purchases, calendars, contacten) en geen user interactie (renderen van bepaalde views, GUI). Er is een uitzondering voor generieke rendering functionaliteit.
- Performance testen met XCTest of aan het einde van het project Xcode profiler gebruiken
  Gekozen om kritieke methodes, zoals het ophalen van calendars en contacten, via XCTest EN Xcode profiler te testen en alle andere functionaliteit via Xcode profiler. In eerste instantie was mijn plan om alles via XCTest te testen maar mijn afstudeer mentor adviseerde dit niet te doen (tijd besparing en niet nodig). Xcode profiler geeft namelijk goede inzichten over performance en wat er knelt.
- Continue integratie in obj-c app of bij voltooiing framework volledig integreren
  Gekozen voor integratie in obj-c bij voltooiing van het framework. Dit motivatie hiervoor was omdat het integreren van het framework in de obj-c tijd kost waardoor het framework veel meer tijd kost om te ontwikkelen. Het is beter om een volledig niet geïntegreerd product te hebben dan een half geïntegreerd product.
- Swift 3
  Later dit jaar (herfst) wordt Swift 3 uitgegeven door Apple, hierbij worden grote veranderingen gemaakt die met zekerheid het framework zullen stuk maken. Er is echter al een beta van Swift 3 beschikbaar en het framework zou al op Swift 3 kunnen worden aangepast. Er is gekozen om wel rekening te houden met Swift 3 en zo veel mogelijk de

Swift 3 standaarden aan te houden. Echter komt de definitieve versie van Swift 3 niet binnen mijn afstudeer termijn uit en valt het dus buiten de scope van het project. De Swift 3 beta zal dus niet gebruikt worden.

- Al bestaande Obj-C code porteren of volledig herschrijven

  Deze keuze zal per functionaliteit worden besloten. Indien een bepaalde functionaliteit nog volgens de nieuwste methode werkt en van goede kwaliteit is, zal deze worden geporteerd naar Swift. Hierbij zullen mogelijk (afhankelijk van Obj-C code) aanpassingen komen om het Swift-compatible te maken.
  Indien een functie een oude standaard gebruik zal deze volledig worden herschreven met de nieuwe standaard. Hoewel de functionaliteit wordt herschreven, moet deze nog altijd aan dezelfde functionaliteit voldoen.

Design beslissingen

- Aggregatie of inheritance (overerving)
  - Swift classes kunnen niet worden overerft door obj-c classes wanneer de swift classes zich binnen een dynamic framework bevinden. Dit is het geval bij dit project waardoor inheritance afvalt.
  - Een groot deel van de iOS SDK is gebouwd met het delegate principe. Dit is een vorm van aggregatie en zal ook goed werken binnen het framework. Een framework vraagt namelijk voornamelijk om implementaties aan de gebruiker.

- Extensions
  - In obj-c en swift is het mogelijk om een class te extenden en er functies en nested classes/structs aan toe te voegen.
  - Hiermee kunnen gemakkelijk base classes zoals String, Date en dergelijken kunnen worden uitgebreid, zonder dat er binnen het project altijd een subtype van deze gebruikt hoeft te worden.

- Pure Swift en ObjC
  - Niet alles dat mogelijk is in Swift kan worden gebridged naar ObjC. Dit betekent dat bepaalde functies binnen het framework niet beschikbaar zijn voor ObjC gebruikers van het framework. Hieronder valt bijvoorbeeld Optional Bool (Bool?), default parameters (func a(a : Int = 0) {} a();//a=0)
  - Er is gekozen om deze niet bridgebare functies zo min mogelijk te gebruiken waar door beschikbaar moeten zijn in ObjC.
  - Een andere oplossing zou zijn om een extra laag tussen Swift en ObjC te bouwen die een brug maakt tussen niet bridgebare code. Dit is echter extra werk en maakt het framework zwaarder en moeilijker om te begrijpen.

- NS(*) vs (*)

  Er zijn meerdere soorten classen die van ObjC naar Swift zijn veranderd, hieronder vallen:
  - NSString->String
  - NSArray->[Type]
  - NSDictionary->[Type, Type]
  - NSInt, NSUInt->Int, UInt

  Alle NS varianten zijn binnen Swift nog altijd beschikbaar
  Conversies tussen NS en * zijn niet zwaar en kunnen gemakkelijk worden gemaakt.

- Runtime flags VS Compile time flags
  - Er kan gekozen worden om ofwel compile time flags te gebruiken, dit betekent dat er statische constanten worden gedefinieerd zoals DEBUG_MODE. Deze kan dan true of false worden gezet door de framework gebruiker en mag niet worden vergeten wanneer er een release word gebuild.
  - De andere optie is runtime flags. Dit zijn opties die in build configuraties kunnen worden gezet. Hierdoor kan het framework in een release build b.v onmogelijk in een debugging state komen (mits de build configuratie correct is)
  - Tot slot is het mogelijk om een hybride hiervan te maken. Dit zal echter de complexiteit van het framework verhogen en is daarom een slechte keuze.
  - Er is gekozen voor runtime flags, hier werd voor het framework al gebruik van gemaakt en is dus al goed bekend bij de gebruikers van het framework.

- NSUserDefaults VS Core Data
  Core Data is vooral bedoeld voor grote data collecties. Core Data geeft veel extra onderhoud werk, maar is wel nuttig wanneer data moet worden geüpdatet naar een ander formaat door de het migratie systeem
  NSUserDefaults daarentegen is speciaal ontwikkeld voor kleine instellingen bij te houden. Door niet teveel op te slaan in NSUserDefaults blijft het snel (4 nanoseconde read c.a). Om deze reden moet er gebruik gemaakt worden van NSUserDefaults

Hieronder staat een opsomming van problemen die zich hebben voorgedaan
- In het begin van het project niet generiek genoeg gewerkt en daardoor app specifieke logica binnen het framework. Hier kwam ik later pas achter. Dit probleem is opgelost door veel code te herschrijven.
- In eerste instantie vaak globale functies gebruikt, echter konden deze niet worden gebruikt voor de integratie van het framework in de obj-c app. Dit moest volledig opnieuw worden gemaakt.
- Veel app specifiek settings logica in het framework gebouwd. Echter was dit niet generiek en dus niet de bedoeling. Dit probleem is opgelost door middel van delegaties.

2.4. Nog te doen

Naast de resterende 30% van de nog te voltooien requirements moet ook het framework nog worden geïmplementeerd in de obj-c app (iPhone app).
De resterende 30% requirements bestaat uit datum formatting en contacten. Deze week staat formatting aan bod.

Het implementeren van het framework in de obj-c app zal veel werk zijn en ook nog onvoorziene problemen met zich meebrengen; echter zijn deze problemen wel geminimaliseerd door het constante gebruik van unit tests. Zonder het constant unit testen van functionaliteit zou de integratie wellicht niet af komen, dit is dus een goede beslissing geweest.
Aanpassingen die alsnog gemaakt moeten worden zullen worden doorgevoerd in het ontwerp en ook weer getest worden nadat deze gebouwd zijn.

3. Conclusie

De hoeveelheid werk dat nog gedaan moet worden is redelijk groot maar wel op schema. Indien er geen grote problemen voorkomen zal de ontwikkeling en integratie volgens planning lopen. Op dit moment ben ik dus positief over het project en verwacht dat het op schema voltooid kan worden. De projectmanagement- en ontwikkeling methodes hebben goed uitgepakt; ze helpen het project stabiel en op schema te houden. De problemen die zich hebben voorgedaan zijn van kleine schaal geweest en konden altijd goed worden opgelost.

7. Deprecation Modernisation rapport

Deprecation Modernisation Rapport

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum **:** 13 Juli 2016
Status **: N/A**
Versie **: 1.0**

1. Inleiding

In dit document wordt beschreven welke functies en code deprecated was en gemoderniseerd is. Deprecated betekend dat er een nieuwe methode is om een bepaalde functie te realiseren. Wanneer een nieuwe methode beschikbaar is zal de oude methode niet meer worden geüpdatet door Apple. Om deze reden is het van belang dat alle deprecated code wordt omgezet naar de meest moderne code.
Er zal binnen dit document eerst een tabel met deprecated functies worden afgebeeld en tot slot een kort rapport over overige semantische- en syntactische fouten.

Het doel van dit document is het duidelijk maken welke deprecations zijn opgelost en dit overzichtelijk houden.

2. Deprecations

Hieronder is een tabel met alle deprecated code die is gemoderniseerd tijdens de loop van het project. Ieder nummer staat voor een bepaalde functie, framework of variabele die niet meer supported is door iOS. Naast het feit dat deprecated code behouden ongewenst is, heeft Apple bij de transitie van Objective-C naar Swift geen deprecated functies meegenomen. Dit betekent dat deprecated functies niet kunnen worden omgezet naar Swift zonder deze te moderniseren.

Tijdens het ontwikkelen van gemoderniseerde code zijn gemoderniseerde functies geschreven in het WeekCal Framework met Swift. Er zijn enkele gevallen waar dit niet is gedaan omdat ofwel de deprecated code van expliciet naar impliciet is gegaan of de modernisering slechts 1 lijn code is waar verder niets mee gedaan wordt.

Nr	Deprecated Methode	Moderne Methode	Notities	Framework implemenatie	Hoeveelheid
1	UIAlert	UIAlertController.Alert	UIAlertController heeft support voor app tint's en popover.	AlertCreater	48
2	UIActionSheet	UIAlertController.ActionSheet	UIAlertController heeft support voor app tint's en popover.	AlertCreater	6
3	device.interfaceOrientation	statusBarOrientation	StatusBarOrientation haalt de oriëntatie van de status bar en niet die van het device. Dit is dus betrouwbaarder. (interface kan portrait zijn terwijl device landscape is)	OrientationUtil	27
4	NSString.drawInRect	String.drawInRect(attributes)	De nieuwe methode heeft support voor decorated strings en is sneller	RenderUtil	47
5	NSString.sizeWithFont	String.sizeWithAttributes	De nieuwe methode heeft support voor decorated strings en is sneller	RenderUtil NSStringDrawing	32

Nr	Deprecated Methode	Moderne Methode	Notities	Framework implemenatie	Hoeveelheid
6	Address Book Framework	Contacts Framework	Waarschijnlijk de grootste verbetering. Contacts werkt volledig object georiënteerd zonder manual pointers in tegenstelling tot Address Book. Is veel sneller en gebruikt minder RAM	ContactsHelper ContactPickerDelegate ContactsHelperDelegate InternalContactsPicker ContactsWrapper ContactsWrapperDelegate LinkedPerson LinkedPersonManager	385
8	NSURLConnection	NSURLSession	NSURLSession heeft geautomatiseerd thread pooling, shared sessions, background support, betere syntax	NSURLSessionExt AveHttpFetcher Reachability Download	10
9	NSString.stringByAppendingPercentEncoding	String.stringByAddingPercentEncodingWithAllowedCharacters	Nieuwe methode heeft support voor character sets zoals: QueryAllowed, PathAllowed, HostAllowed voor betrouwbare URL Encoding	NSStringExt	17
10	NSData.base64String	NSData.base64EncodedStringWithOptions	Nieuwe methode heeft support voor verschillende Base64 Encoding types	NSDataExt	4
11	WeekCal Emoji	NSStringEmoji	Oude methode van WeekCal was outdated. De nieuwe door mij gebouwde methode detecteert Emoji betrouwbaarder en uitbreidbaarder.	NSStringEmoji	1
12	inout, pointer parameters	Tuples	De oude methode gebruikte pointers in parameters om de parameters te updaten van de methode caller. Dit is onveilig en niet thread secure. De nieuwe methode return tuples (arrays)	Veel Onder andere UIColor	

Nr	Deprecated Methode	Moderne Methode	Notities	Framework implemenatie	Hoeveelheid
13	UISeachDisplayController	UISeachController	De nieuwe methode heeft support voor implicit table views, segmented controls, betere delegates en algemene verbeteringen	SearchCreater. Is echter niet volledig geimplementeerd omdat dit design veranderingen nodig heeft.	6
14	NDateComponents.week	NSDateComponents.weekOfYear	Oude variabele wordt niet altijd meer supported. Nieuwe wel en is dus betrouwbaarder	NSDateComponentsExt	3
15	UISegementedControlerStyleBar	Geen	Oude methode wordt niet meer gebruikt. Nieuwe methode is er niet, oude methode is impliciet geworden.		18
16	NSGregorianCalendar	NSCalendarIdentifier.Gregorian	Betere syntax	NSDateComponentsExt DateUtil BaseDateUtil NSDateExt	1
17	NSData.initWithBase64Encoding	NSData.initWithBase64EncodedString(str, options)	Betere benaming, heeft mogelijkheid om base64 encoding type te kiezen	NSDataExt	1
18	CLRegion.radius & CLRegion.center	CLCircularRegion.radius & center	Betere verantwoordelijkheid	LocationManager	4
19	UIBarButtonItemStyleBordered	UIBarButtonItemStylePlain	Nieuwe methode doet wat oude doet en wordt door meer UI's ondersteund		9
20	dismissModelViewController	dismissViewController	Modal view controllers zijn niet langer supported. De oude methode was langzamer		1
21	contentSizeForViewInPopover	preferredContentSize	preferredContentSize is meer gecentraliseerd.	ViewUtil	
22	UILineBreakMode	NSLineBreakMode	Support meer UI's	ViewUtil	1
23	kCTText(X)Alignment	kCTTextAlignment(X)	Support meer UI's	ViewUtil	3
24	EKDay	EKWeekDay	Betere Support		7

Nr	Deprecated Methode	Moderne Methode	Notities	Framework implemenatie	Hoeveelheid
25	setStatusBarStyle		Functie is verwijderd. Kan evt. preferredStatusBarStyle override in viewController.		3
	Totaal				**631**

3. Overige Semantische- en Syntactische fouten

Naast deprecations bestaan er ook onveilige of gewoon fout geschreven code in de huidige WeekCal app. Hieronder wordt beschreven welke fouten er zijn gevonden en hoe deze zijn opgelost

Fout	Oplossing	Hoeveelheid
Expliciete Selector naar functies die alleen bestaan in een subklasse van het target van de selector. BV saveEvent op UIViewController waar de functie alleen bestaat in EditEventViewController, subklasse van UIViewController. De code zal werken omdat de selector alleen kan worden uitgevoerd wanneer de functie bestaat maar is niet goed leesbaar en kan in latere releases problemen geven	Omgezet naar impliciete Selectors. Hierdoor weet de Selector dat de functie misschien niet bestaat en kan deze zich hierop aanpassen.	5
Property synthesis van onbestaande properties. Dit betekend dat een getter en setter automatisch wordt gemaakt. Echter bestaat in dit geval de property in de subklasse, maar wel in de superklasse. Dit kan onverwacht gedrag veroorzaken	In plaats van @synthesize, @dynamic gebruikt. Hierdoor weet iOS dat de property mogelijk in een superklasse kan bestaan.	3
Null verstuurd naar NON NULL parameter. Dit kan BAD_ACCESS errors geven	Dummy waarde verstuurd.	1
Unused Variables. Nutteloze variabelen, kan memory leak opleveren en maakt leesbaarheid van code slechter	Ongebruikte variabelen verwijdert.	18
Conflicting Parameters Verschillende Functies hebben dezelfde parameters en naam. Kan onverwacht gedrag en fouten veroorzaken	Methode refactoren	1
Totaal		28

8. Test Rapportage Juli

Test Rapportage 11-07-2016

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum : 11 Juli 2016
Status : **Final**
Versie : **1.0**

1. Inleiding

In dit rapport worden de bevindingen van de testrun die op maandag 11 juli is uitgevoerd vastgelegd. De resultaten bestaan uit testen uitkomsten, performance en code coverage. Tijdens de duur van het project zullen er nog meer test rapportages worden opgesteld naarmate meer testen ontstaan.

2. Test resultaten

In dit hoofdstuk worden de test resultaten statistisch beschreven en vervolgens afgebeeld. De betreffende testen zijn unit-tests die binnen het project zijn geïntegreerd via de iOS ontwikkeling tool Xcode.

Statistiek Testrun

Hoeveelheid Tests	288
Waarvan functioneel	283
Waarvan performance	5
Hoeveelheid geslaagde tests	288
Hoeveelheid gefaalde tests	0
Percentage geslaagd	100%
Test Omgeving	Xcode 7 OS X El Capitan iPhone 6S Engelse Taal

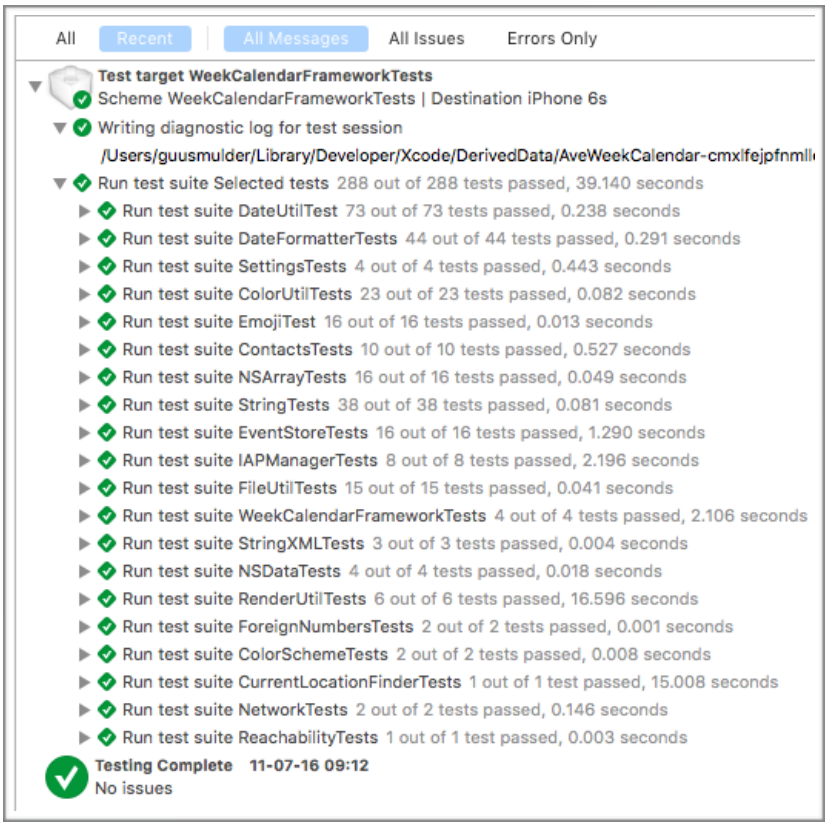
Tabel 1

Zoals in tabel 1 is te zien, is 100% van de tests geslaagd. Doordat tijdens het ontwikkelen van het WeekCal framework er gebruik wordt gemaakt van de ontwikkelmethode test-driven development is dit resultaat mogelijk gemaakt. Iedere functie die wordt gebouwd is namelijk gebaseerd op een unit test en wordt doorontwikkelde totdat zijn unit test slaagt.
Om deze reden zouden test rapportages minder nuttig zijn maar dit is niet het geval. Iedere unit test wordt apart uitgevoerd tijdens test-driven development in tegenstelling tot de volledige testrun voor de testrapportage. Op deze manier worden testen die mogelijkheid hebben tot het beïnvloeden van elkaar (koppeling) ook volledig getest.

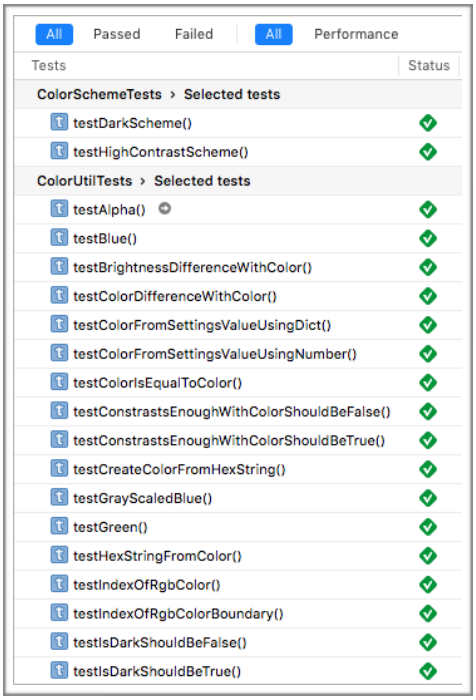
Test resultaten afgebeeld

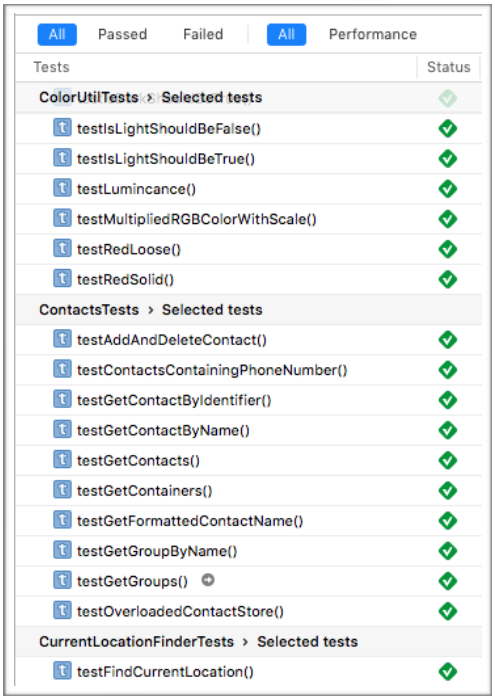
Functionele tests slagen of falen gebaseerd op een expectation-result systeem. Xcode vraagt om een expected waarde. Vervolgens vergelijkt Xcode of expected gelijk is met result van de test. Indie deze gelijk zijn slaagt de test; indien deze niet gelijk zijn faalt de test.

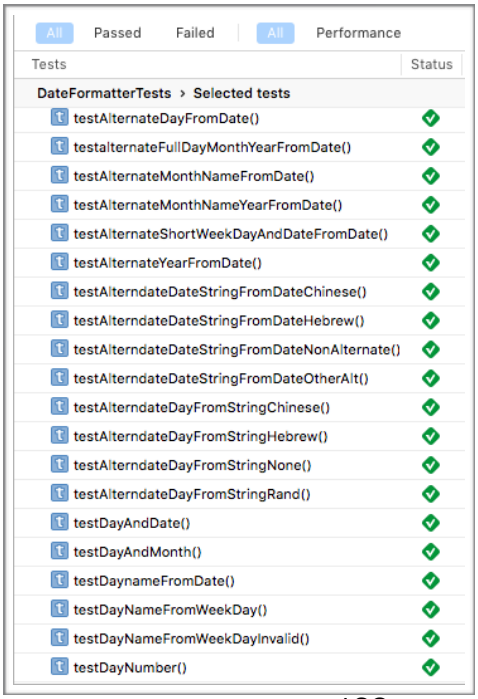
Samengevat

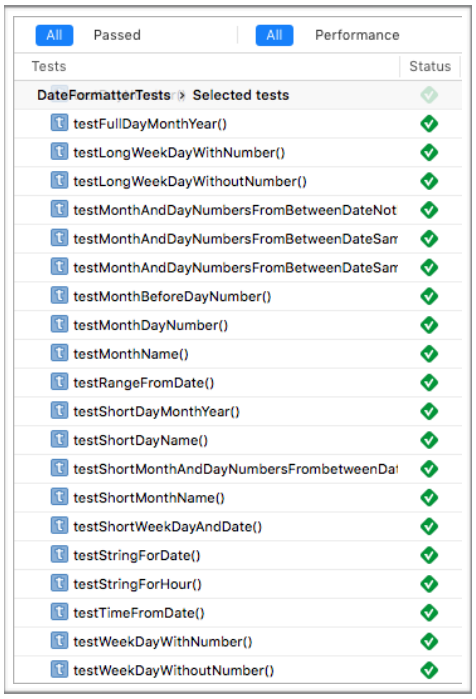


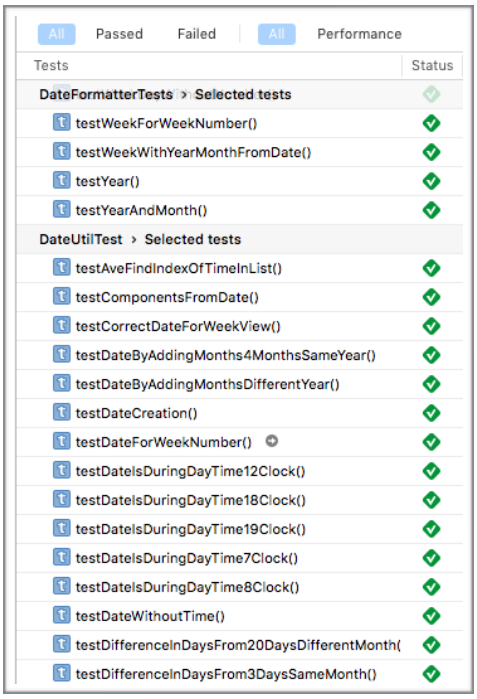
Functioneel & Performance

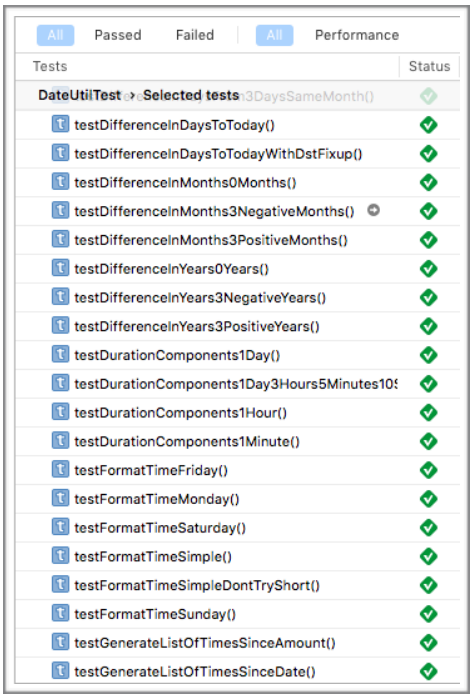


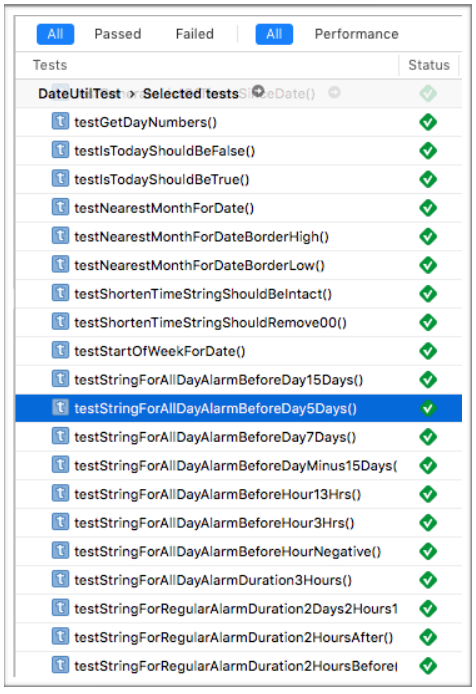


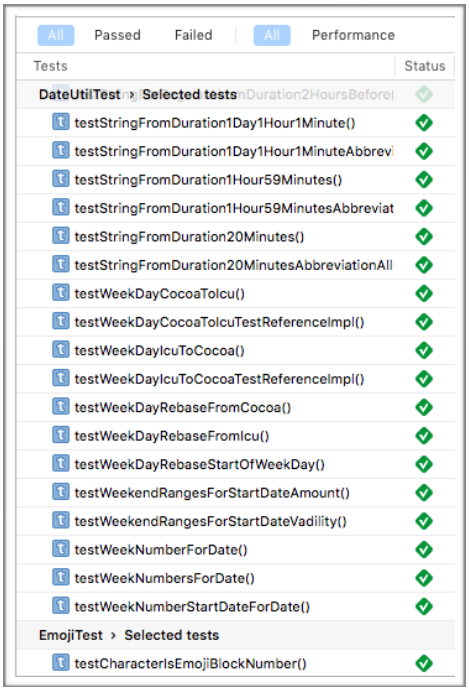


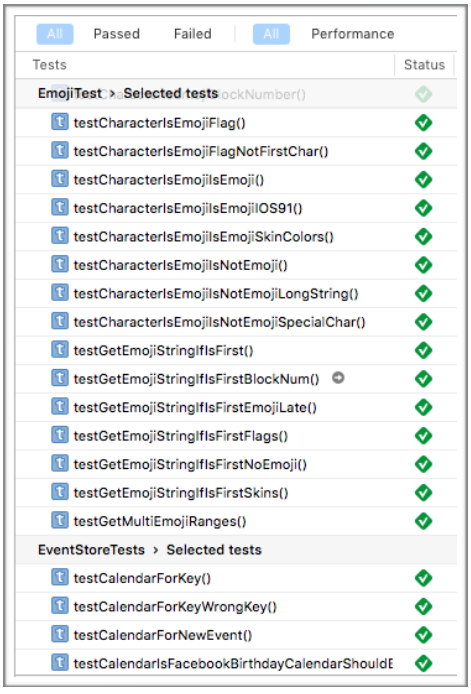

All
Passed
All
Performance
Tests
Status
DateFormatterTests › Selected tests
testFullDayMonthYear()
testLongWeekDayWithNumber()
testLongWeekDayWithoutNumber()
testMonthAndDayNumbersFromBetweenDateNot
testMonthAndDayNumbersFromBetweenDateSan
testMonthAndDayNumbersFromBetweenDateSan
testMonthBeforeDayNumber()
testMonthDayNumber()
testMonthName()
testRangeFromDate()
testShortDayMonthYear()
testShortDayName()
testShortMonthAndDayNumbersFrombetweenDa
testShortMonthName()
testShortWeekDayAndDate()
testStringForDate()
testStringForHour()
testTimeFromDate()
testWeekDayWithNumber()
testWeekDayWithoutNumber()

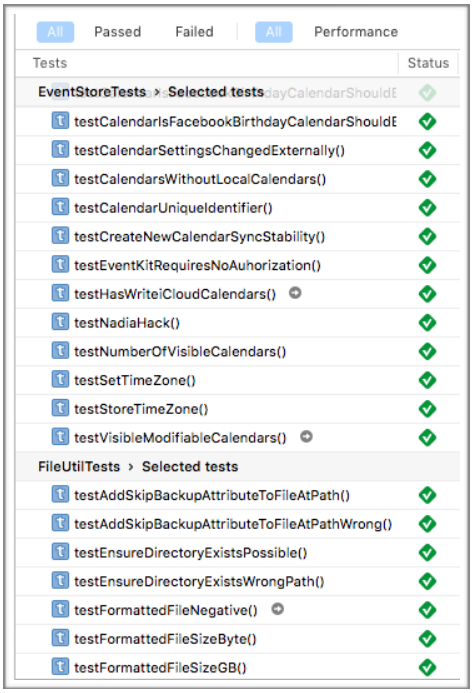

All
Passed
Failed
All
Performance
Tests
Status
DateFormatterTests › Selected tests
testWeekForWeekNumber()
testWeekWithYearMonthFromDate()
testYear()
testYearAndMonth()
DateUtilTest › Selected tests
testAveFindIndexOfTimeInList()
testComponentsFromDate()
testCorrectDateForWeekView()
testDateByAddingMonths4MonthsSameYear()
testDateByAddingMonthsDifferentYear()
testDateCreation()
testDateForWeekNumber()
testDateIsDuringDayTime12Clock()
testDateIsDuringDayTime18Clock()
testDateIsDuringDayTime19Clock()
testDateIsDuringDayTime7Clock()
testDateIsDuringDayTime8Clock()
testDateWithoutTime()
testDifferenceInDaysFrom20DaysDifferentMonth(
testDifferenceInDaysFrom3DaysSameMonth()

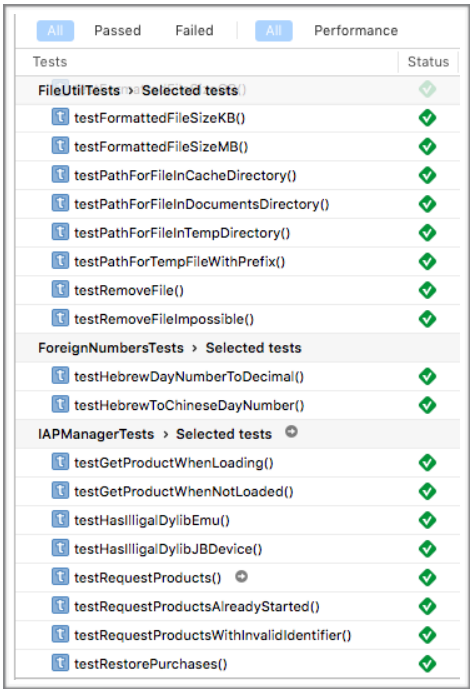

All
Passed
Failed
All
Performance
Tests
Status
DateUtilTest › Selected tests
testDifferenceInDaysToToday()
testDifferenceInDaysToTodayWithDstFixup()
testDifferenceInMonths0Months()
testDifferenceInMonths3NegativeMonths()
testDifferenceInMonths3PositiveMonths()
testDifferenceInYears0Years()
testDifferenceInYears3NegativeYears()
testDifferenceInYears3PositiveYears()
testDurationComponents1Day()
testDurationComponents1Day3Hours5Minutes10
testDurationComponents1Hour()
testDurationComponents1Minute()
testFormatTimeFriday()
testFormatTimeMonday()
testFormatTimeSaturday()
testFormatTimeSimple()
testFormatTimeSimpleDontTryShort()
testFormatTimeSunday()
testGenerateListOfTimesSinceAmount()
testGenerateListOfTimesSinceDate()

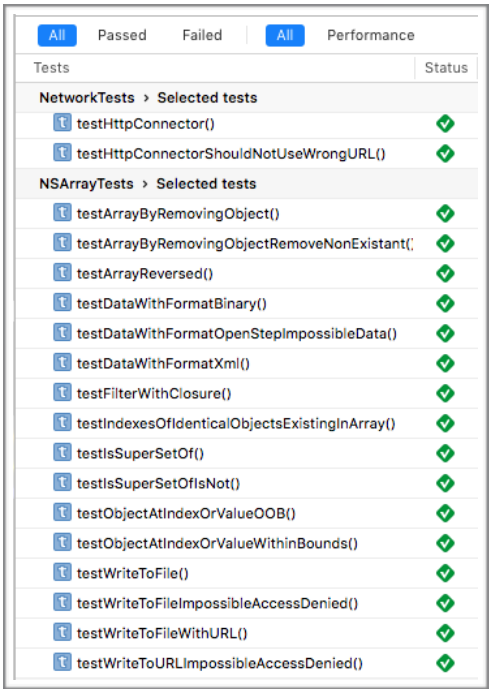

All
Passed
Failed
All
Performance
Tests
Status
DateUtilTest › Selected tests
testGetDayNumbers()
testIsTodayShouldBeFalse()
testIsTodayShouldBeTrue()
testNearestMonthForDate()
testNearestMonthForDateBorderHigh()
testNearestMonthForDateBorderLow()
testShortenTimeStringShouldBeIntact()
testShortenTimeStringShouldRemove00()
testStartOfWeekForDate()
testStringForAllDayAlarmBeforeDay15Days()
testStringForAllDayAlarmBeforeDay5Days()
testStringForAllDayAlarmBeforeDay7Days()
testStringForAllDayAlarmBeforeDayMinus15Days(
testStringForAllDayAlarmBeforeHour13Hrs()
testStringForAllDayAlarmBeforeHour3Hrs()
testStringForAllDayAlarmBeforeHourNegative()
testStringForAllDayAlarmDuration3Hours()
testStringForRegularAlarmDuration2Days2Hours1
testStringForRegularAlarmDuration2HoursAfter()
testStringForRegularAlarmDuration2HoursBefore

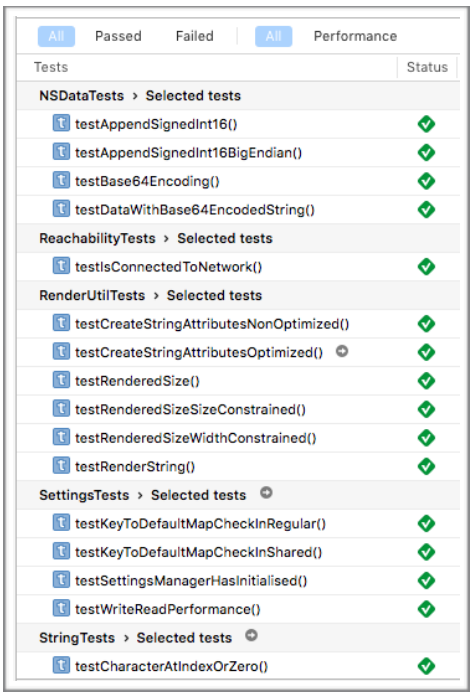

All
Passed
Failed
All
Performance
Tests
Status
DateUtilTest › Selected tests
testStringFromDuration1Day1Hour1Minute()
testStringFromDuration1Day1Hour1MinuteAbbrev
testStringFromDuration1Hour59Minutes()
testStringFromDuration1Hour59MinutesAbbreviat
testStringFromDuration20Minutes()
testStringFromDuration20MinutesAbbreviationAll
testWeekDayCocoaTolcu()
testWeekDayCocoaTolcuTestReferenceImpl()
testWeekDayIcuToCocoa()
testWeekDayIcuToCocoaTestReferenceImpl()
testWeekDayRebaseFromCocoa()
testWeekDayRebaseFromIcu()
testWeekDayRebaseStartOfWeekDay()
testWeekendRangesForStartDateAmount()
testWeekendRangesForStartDateVadility()
testWeekNumberForDate()
testWeekNumbersForDate()
testWeekNumberStartDateForDate()
EmojiTest › Selected tests
testCharacterIsEmojiBlockNumber()

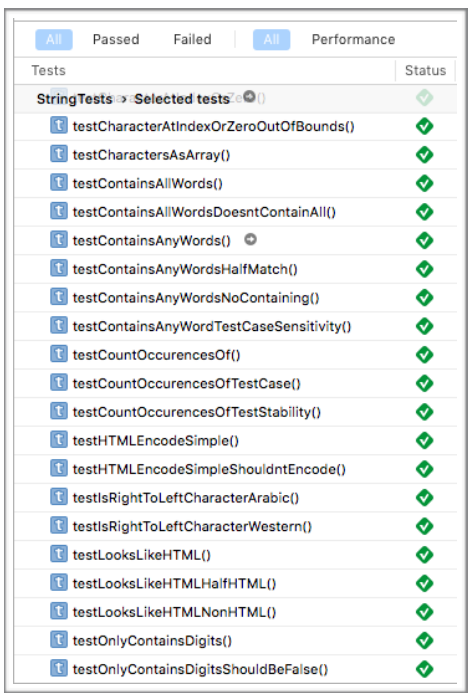

All
Passed
Failed
All
Performance
Tests
Status
EmojiTest › Selected tests
testCharacterIsEmojiFlag()
testCharacterIsEmojiFlagNotFirstChar()
testCharacterIsEmojiIsEmoji()
testCharacterIsEmojiIsEmojiIOS91()
testCharacterIsEmojiIsEmojiSkinColors()
testCharacterIsEmojiIsNotEmoji()
testCharacterIsEmojiIsNotEmojiLongString()
testCharacterIsEmojiIsNotEmojiSpecialChar()
testGetEmojiStringIfIsFirst()
testGetEmojiStringIfIsFirstBlockNum()
testGetEmojiStringIfIsFirstEmojiLate()
testGetEmojiStringIfIsFirstFlags()
testGetEmojiStringIfIsFirstNoEmoji()
testGetEmojiStringIfIsFirstSkins()
testGetMultiEmojiRanges()
EventStoreTests › Selected tests
testCalendarForKey()
testCalendarForKeyWrongKey()
testCalendarForNewEvent()
testCalendarIsFacebookBirthdayCalendarShouldE

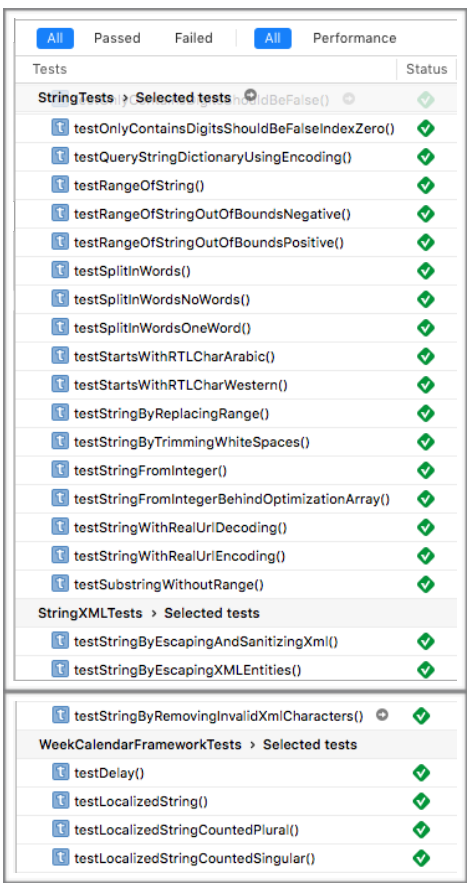

All
Passed
Failed
All
Performance
Tests
Status
EventStoreTests › Selected tests
testCalendarIsFacebookBirthdayCalendarShouldE
testCalendarSettingsChangedExternally()
testCalendarsWithoutLocalCalendars()
testCalendarUniqueIdentifier()
testCreateNewCalendarSyncStability()
testEventKitRequiresNoAuhorization()
testHasWriteiCloudCalendars()
testNadiaHack()
testNumberOfVisibleCalendars()
testSetTimeZone()
testStoreTimeZone()
testVisibleModifiableCalendars()
FileUtilTests › Selected tests
testAddSkipBackupAttributeToFileAtPath()
testAddSkipBackupAttributeToFileAtPathWrong()
testEnsureDirectoryExistsPossible()
testEnsureDirectoryExistsWrongPath()
testFormattedFileNegative()
testFormattedFileSizeByte()
testFormattedFileSizeGB()


All
Passed
Failed
All
Performance
Tests
Status
FileUtilTests › Selected tests
testFormattedFileSizeKB()
testFormattedFileSizeMB()
testPathForFileInCacheDirectory()
testPathForFileInDocumentsDirectory()
testPathForFileInTempDirectory()
testPathForTempFileWithPrefix()
testRemoveFile()
testRemoveFileImpossible()
ForeignNumbersTests › Selected tests
testHebrewDayNumberToDecimal()
testHebrewToChineseDayNumber()
IAPManagerTests › Selected tests
testGetProductWhenLoading()
testGetProductWhenNotLoaded()
testHasIllegalDylibEmu()
testHasIllegalDylibJBDevice()
testRequestProducts()
testRequestProductsAlreadyStarted()
testRequestProductsWithInvalidIdentifier()
testRestorePurchases()

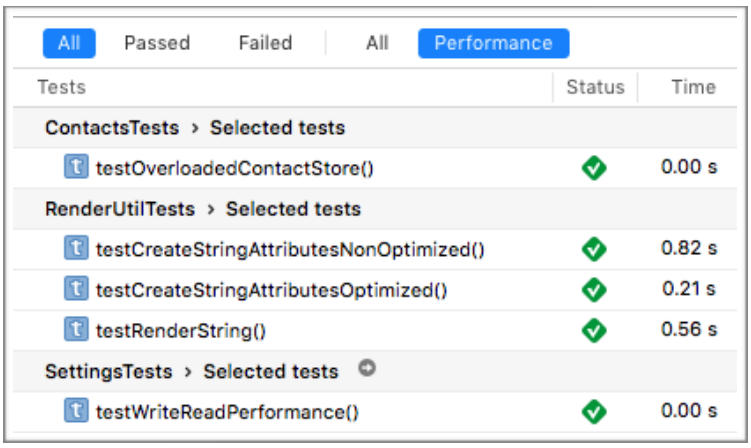

All
Passed
Failed
All
Performance
Tests
Status
NetworkTests › Selected tests
testHttpConnector()
testHttpConnectorShouldNotUseWrongURL()
NSArrayTests › Selected tests
testArrayByRemovingObject()
testArrayByRemovingObjectRemoveNonExistant(
testArrayReversed()
testDataWithFormatBinary()
testDataWithFormatOpenStepImpossibleData()
testDataWithFormatXml()
testFilterWithClosure()
testIndexesOfIdenticalObjectsExistingInArray()
testIsSuperSetOf()
testIsSuperSetOfIsNot()
testObjectAtIndexOrValueOOB()
testObjectAtIndexOrValueWithinBounds()
testWriteToFile()
testWriteToFileImpossibleAccessDenied()
testWriteToFileWithURL()
testWriteToURLImpossibleAccessDenied()


All
Passed
Failed
All
Performance
Tests
Status
NSDataTests › Selected tests
testAppendSignedInt16()
testAppendSignedInt16BigEndian()
testBase64Encoding()
testDataWithBase64EncodedString()
ReachabilityTests › Selected tests
testIsConnectedToNetwork()
RenderUtilTests › Selected tests
testCreateStringAttributesNonOptimized()
testCreateStringAttributesOptimized()
testRenderedSize()
testRenderedSizeSizeConstrained()
testRenderedSizeWidthConstrained()
testRenderString()
SettingsTests › Selected tests
testKeyToDefaultMapCheckInRegular()
testKeyToDefaultMapCheckInShared()
testSettingsManagerHasInitialised()
testWriteReadPerformance()
StringTests › Selected tests
testCharacterAtIndexOrZero()


All
Passed
Failed
All
Performance
Tests
Status
StringTests › Selected tests
testCharacterAtIndexOrZeroOutOfBounds()
testCharactersAsArray()
testContainsAllWords()
testContainsAllWordsDoesntContainAll()
testContainsAnyWords()
testContainsAnyWordsHalfMatch()
testContainsAnyWordsNoContaining()
testContainsAnyWordTestCaseSensitivity()
testCountOccurencesOf()
testCountOccurencesOfTestCase()
testCountOccurencesOfTestStability()
testHTMLEncodeSimple()
testHTMLEncodeSimpleShouldntEncode()
testIsRightToLeftCharacterArabic()
testIsRightToLeftCharacterWestern()
testLooksLikeHTML()
testLooksLikeHTMLHalfHTML()
testLooksLikeHTMLNonHTML()
testOnlyContainsDigits()
testOnlyContainsDigitsShouldBeFalse()


All
Passed
Failed
All
Performance
Tests
Status
StringTests › Selected tests
testOnlyContainsDigitsShouldBeFalseIndexZero()
testQueryStringDictionaryUsingEncoding()
testRangeOfString()
testRangeOfStringOutOfBoundsNegative()
testRangeOfStringOutOfBoundsPositive()
testSplitInWords()
testSplitInWordsNoWords()
testSplitInWordsOneWord()
testStartsWithRTLCharArabic()
testStartsWithRTLCharWestern()
testStringByReplacingRange()
testStringByTrimmingWhiteSpaces()
testStringFromInteger()
testStringFromIntegerBehindOptimizationArray()
testStringWithRealUrlDecoding()
testStringWithRealUrlEncoding()
testSubstringWithoutRange()
StringXMLTests › Selected tests
testStringByEscapingAndSanitizingXml()
testStringByEscapingXMLEntities()
testStringByRemovingInvalidXmlCharacters()
WeekCalendarFrameworkTests › Selected tests
testDelay()
testLocalizedString()
testLocalizedStringCountedPlural()
testLocalizedStringCountedSingular()

Performance Specifiek



Performance tests slagen of falen door na te gaan hoe lang de test bezig is geweest. Xcode vraagt aan de tester hoe lang de test maximaal mag duren, indien vervolgens de test langer duurt dan aangegeven faalt hij.

3. Code coverage

Code coverage binnen XCode is het percentage code binnen het project dat ge-unit test is. Het percentage zal nooit zeer hoog liggen omdat veel code niet getest kan worden. Deze ontestbare code bestaat uit situaties die niet gesimuleerd kunnen worden zoals geheugen corruptie, apparaat specifieke problemen en problemen die random ontstaan. Hiernaast wordt code syntax vaak ook niet meegerekend binnen code coverage, maar haalt het wel het percentage naar beneden.

Het verschil tussen de generieke term code coverage, dat de diepte van een test betekend (hoeveel paden zijn afgeschermd) en Xcode code coverage, is dat Xcode de code coverage live meet door bij te houden wat wel en wat niet wordt uitgevoerd door unit tests. Dit geeft een zeer realistisch beeld over de echte code coverage. Echter moet er wel worden uitgegaan van een foutmarge van circa 10%.

Tijdens dit project wordt er gestreefd naar een code coverage van 60% binnen het framework. Hoger is beter, het laagste percentage dat acceptabel is, is 50%. Het is echter lastig om het percentage hoog te houden doordat er steeds meer complexere functionaliteit wordt ingebouwd met diepere paden. Hierbij moet de keuze worden gemaakt tussen tijd besteden aan hogere code coverage of meer functionaliteit. Tot nu toe is deze keuze verdeeld geweest, bij tijdgebrek werd er vaak gekozen voor functionaliteit en vice versa.

Code coverage statistiek

Framework Code Coverage	58%
ObjC app Code Coverage	16%

Tabel 2

Uit tabel 2 blijkt dat het framework een code coverage heeft van 58% en de ObjC app 16%. De ObjC app is de WeekCal app die het framework gebruikt.
Op dit moment is nog niet het volledige framework geïntegreerd in de WeekCal ObjC app, maar wel al een aantal delen. Hierdoor is de 16% code coverage mogelijk, dit was voorheen namelijk 0%.

Het percentage van 58 bij het WeekCal framework is bijna bij het streefdoel van 60%. Er zijn een aantal functies die nog unit tests nodig hebben doordat deze in haast zijn ontwikkeld om een scrum doel te behalen.

9. Test rapportage Augustus

Test Rapportage 16-08-2016

WeekCal Framework

WeekCal

Project : WeekCal Framework

Leden : Guus Mulder

Datum : 16 Augustus 2016
Status : **Final**
Versie : **1.0**

1. Inleiding

In dit rapport worden de bevindingen van de testrun die op dinsdag 16 Augustus is uitgevoerd vastgelegd. De resultaten bestaan uit testen uitkomsten, performance en code coverage. Tijdens de duur van het project zullen er nog meer test rapportages worden opgesteld naarmate meer testen ontstaan. Tot slot wordt er een advies gegeven over het in productie brengen van het WeekCal framework.

2. Test resultaten

In dit hoofdstuk worden de test resultaten statistisch beschreven en vervolgens afgebeeld. De betreffende testen zijn unit-tests die binnen het project zijn geïntegreerd via de iOS ontwikkeling tool Xcode.

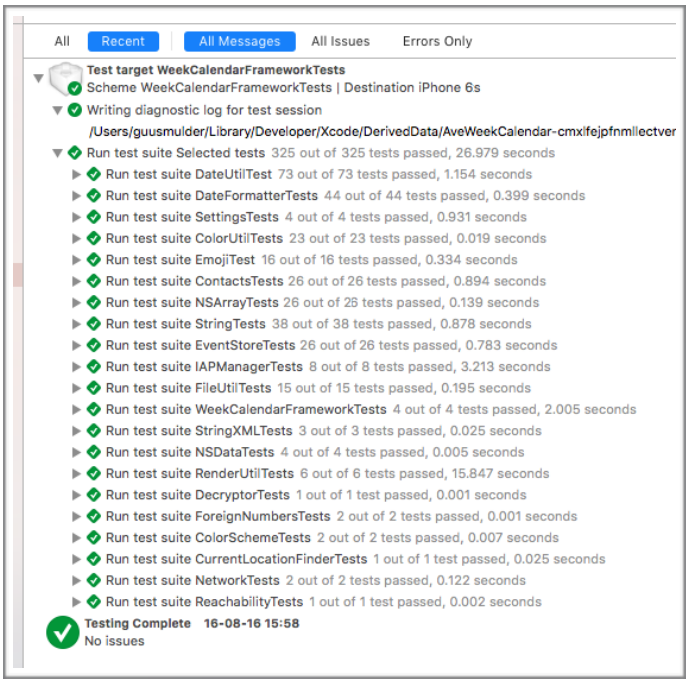
Statistiek Testrun

Hoeveelheid Tests	325
Waarvan functioneel	320
Waarvan performance	5
Hoeveelheid geslaagde tests	325
Hoeveelheid gefaalde tests	0
Percentage geslaagd	100%
Test Omgeving	Xcode 7 OS X El Capitan iPhone 6S Engelse Taal

Tabel 1

Zoals in tabel 1 is te zien, is 100% van de tests geslaagd. Doordat tijdens het ontwikkelen van het WeekCal framework er gebruik wordt gemaakt van de ontwikkelmethode test-driven development is dit resultaat mogelijk gemaakt. Iedere functie die wordt gebouwd is namelijk gebaseerd op een unit test en wordt doorontwikkelde totdat zijn unit test slaagt.
Om deze reden zouden test rapportages minder nuttig zijn maar dit is niet het geval. Iedere unit test wordt apart uitgevoerd tijdens test-driven development in tegenstelling tot de volledige testrun voor de testrapportage. Op deze manier worden testen die mogelijkheid hebben tot het beïnvloeden van elkaar (koppeling) ook volledig getest.

Test resultaten afgebeeld
Functionele tests slagen of falen gebaseerd op een expectation-result systeem. Xcode vraagt om een expected waarde. Vervolgens vergelijkt Xcode of expected gelijk is met result van de test. Indien deze gelijk zijn slaagt de test; indien deze niet gelijk zijn faalt de test.



Testrun Rapport van XCode

3. Code coverage

Code coverage binnen XCode is het percentage code binnen het project dat ge-unit test is. Het percentage zal nooit zeer hoog liggen omdat veel code niet getest kan worden. Deze ontestbare code bestaat uit situaties die niet gesimuleerd kunnen worden zoals geheugen corruptie, apparaat specifieke problemen en problemen die random ontstaan. Hiernaast wordt code syntax vaak ook niet meegerekend binnen code coverage, maar haalt het wel het percentage naar beneden.

Het verschil tussen de generieke term code coverage, dat de diepte van een test betekend (hoeveel paden zijn afgeschermd) en Xcode code coverage, is dat Xcode de code coverage live meet door bij te houden wat wel en wat niet wordt uitgevoerd door unit tests. Dit geeft een zeer realistisch beeld over de echte code coverage. Echter moet er wel worden uitgegaan van een foutmarge van circa 10%.

Tijdens dit project wordt er gestreefd naar een code coverage van 60% binnen het framework. Hoger is beter, het laagste percentage dat acceptabel is, is 50%. Het is echter lastig om het percentage hoog te houden doordat er steeds meer complexere functionaliteit wordt ingebouwd met diepere paden. Hierbij moet de keuze worden gemaakt tussen tijd besteden aan hogere code coverage of meer functionaliteit. Tot nu toe is deze keuze verdeeld geweest, bij tijdgebrek werd er vaak gekozen voor functionaliteit en vice versa.
Code coverage statistiek

Framework Code Coverage	60%
ObjC app Code Coverage	29%

Tabel 2

Uit tabel 2 blijkt dat het framework een code coverage heeft van 56% en de ObjC app 22%. De ObjC app is de WeekCal app die het framework gebruikt. Op dit moment is het WeekCal framework voor circa 80% geïntegreerd in de Week Calendar app. Hierdoor is de 29% code coverage mogelijk, dit was voorheen namelijk 22%.

Het streefdoel 60% code coverage binnen het WeekCal framework is nu bereikt, waar dat in de vorige testrapportage nog bij 58% lag.

4. Productie Advies

Het WeekCal is tijdens deze testrapportage voltooid en voor circa 80% geïntegreerd in de Week Calendar app, met deze cijfers kan ik een oordeel geven over de het in productie brengen van het WeekCal framework. Omdat het WeekCal framework zeer veel veranderingen zal brengen in de huidige code base is het onverstandig om het gehele framework in één release in productie te brengen. Hoewel alle tests slagen en de code dus betrouwbaar is, zijn mobiele apparaten onvoorspelbaar. Zolang het in productie brengen van het WeekCal framework per module wordt gefaseerd kan ik door middel van de testresultaten concluderen dat het risico op problemen laag is. Als zich dan toch problemen voordoen is het direct duidelijk waar deze vandaan komen en kunnen de problemen worden opgelost, bij een volledige release van het WeekCal framework zal het zeer lastig zijn om problemen op te sporen. Ik adviseer om de modules in deze volgorde in productie te brengen:

1. Contact module
2. Extensions module
3. Util module
4. Date Utilities module
5. CoreLocation & EventStore module
6. Theming module
7. Networking module
8. StoreKit module

10. Afstudeerplan

Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2016-1.2 (start uiterlijk 9 mei 2016)
Startdatum uitvoering afstudeeropdracht: 9 mei 2016
Inleverdatum afstudeerdossier volgens jaarrooster: 3 oktober 2016

Studentnummer: 12011029
Achternaam: dhr. Mulder
Voorletters: G
Roepnaam: Guus
Adres: Stuyvesantstraat 322
Postcode: 2593GW
Woonplaats: Den Haag
Telefoonnummer: 06-31231797
Mobiel nummer: 06-31231797
Privé emailadres: guuuss@live.nl

Opleiding: Informatica
Locatie: Den Haag
Variant: voltijd

Naam studieloopbaanbegeleider: Gerda in ‘t Veld
Naam begeleidend examinator: A.J. Toet
Naam tweede examinator: H.J. Middendorp

Naam bedrijf: WeekCal
Afdeling bedrijf: App Development
Bezoekadres bedrijf: Turfschipper 7
Postcode bezoekadres: 2292JC Wateringen
Postbusnummer:
Postcode postbusnummer:
Plaats: Wateringen
Telefoon bedrijf: 088-8822290
Telefax bedrijf:
Internetsite bedrijf: webbeat.nl

Achternaam opdrachtgever: dhr. Garg
Voorletters opdrachtgever: S
Titulatuur opdrachtgever:
Functie opdrachtgever: Lead iOS Developer
Doorkiesnummer opdrachtgever:
Email opdrachtgever: saurabh@weekcal.com

Achternaam bedrijfsmentor: dhr. Garg
Voorletters bedrijfsmentor: S
Titulatuur bedrijfsmentor:
Functie bedrijfsmentor: Lead iOS Developer
Doorkiesnummer bedrijfsmentor:
Email bedrijfsmentor: saurabh@weekcal.com

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal): iOS Developer

Titel afstudeeropdracht:

Ontwikkelen van een gemeenschappelijk framework voor de applicatie Week Calendar bij WeekCal.

Opdrachtomschrijving

1. Bedrijf

WeekCal is een bedrijf dat zich richt op het ontwikkelen en publiceren van mobiele applicaties. Een van de grootste en meest succesvolle mobiele applicaties van WeekCal is de applicatie 'Week Calendar'. Week Calendar is een agenda applicatie die momenteel op nummer 2 staat in de Apple app store onder de categorie Productiviteit (betaald). De primaire doelgroep bestaat uit smartphone eigenaars die niet tevreden zijn met de standaard agenda in iOS en Android. De Week Calendar app heeft wereldwijd ruim een miljoen zeer actieve gebruikers.
De afdeling waar mijn opdracht zal worden uitgevoerd is de iOS app development afdeling. Binnen deze afdeling worden nieuwe iOS applicaties en updates voor al bestaande iOS applicaties ontwikkeld en gepubliceerd, de tegenhanger hiervan is de Android afdeling, waar ditzelfde wordt gedaan voor de Android variant van de Week Calendar applicatie. De iOS afdeling bestaat uit circa 15 ontwikkelaars die samen op een verdieping binnen het gebouw werken. Omdat de ontwikkelaars in dezelfde ruimte werken ontstaat er een informele sfeer waar het mogelijk is om gemakkelijk vragen te stellen of advies te vragen aan andere werknemers.

2. Probleemstelling

WeekCal heeft op dit moment twee verschillende Week Calendar apps in de App Store, voor zowel iPhone als iPad is er namelijk een aparte iOS applicatie ontwikkeld. Beide apps gebruiken zo veel mogelijk dezelfde codebase, beide met specifieke branches in de user interface elementen. De iPhone applicatie heeft meer features en functionaliteiten dan zijn iPad tegenhanger. Ook al gebruiken deze applicaties voor een groot deel dezelfde architectuur, worden tijdens het bouw proces beide projecten als individuele codebases behandeld. Het effect hiervan is dat veranderingen die tijdens het ontwikkelen van één van de apps zijn gemaakt niet automatisch mee veranderen op de andere app.
De huidige oplossing in het delen van de codebase zorgt voor de volgende problemen:
- Gelijktijdige ontwikkeling aan de beide apps is complex (of onmogelijk);
- Het aantal ontwikkelaars wat gelijktijdig aan de apps kan werken wordt beperkt door de snel toenemende complexiteit bij het doorvoeren van aanpassingen in de codebase;
- Nieuwe ontwikkelingen kunnen minder eenvoudig op de achtergrond als beta-feature meegenomen wordt in een normale release;
- Code kan niet hergebruikt worden in andere apps zoals bijvoorbeeld die van Datumprikker.

3. Doelstelling van de afstudeeropdracht

Het doel van dit project is het ontwikkelen van een WeekCal framework dat alle functionaliteit aanbiedt van de al bestaande iPhone en iPad Week Calendar iOS app, m.u.v. de user interfaces. Vervolgens kunnen de iPhone en iPad applicatie beiden het framework gebruiken en dus, op user interface na, hun codebase weghalen. Wanneer er een update/verandering is binnen het framework kunnen beide applicaties direct worden geüpdatet met de nieuwste codebase, in plaats van dat beide applicaties hun codebase moeten aanpassen. Het doel dat hiermee bereikt wordt is het vergemakkelijken van het onderhoud van de iPhone en iPad applicatie. Doordat er slechts één gescheiden codebase als framework is, is consistentie geen probleem meer, kunnen meerdere ontwikkelaars tegelijk eraan werken en kan code gemakkelijk worden hergebruikt door andere applicaties die het framework gebruiken. Door deze verandering kost het onderhouden van de Week Calendar applicaties minder tijd en geld.
Tijdens het ontwikkelen van het WeekCal framework is het de bedoeling dat de functionaliteiten die in het framework worden geplaatst ook direct naar Swift code worden omgezet en wanneer deze functionaliteiten van oude API's gebruik maken, deze moderniseren.

4. Resultaat

1. Een gemoderniseerde code base, obsolete of verouderde (deprecated) code is omgezet naar moderne code; dit maakt de applicatie stabieler en betrouwbaarder.
2. Het gemakkelijk gemaakt om zowel de iPhone als iPad versies (bijna) tegelijk te releasen bij updates.

3. WeekCal framework gemigreerd van Objective-C naar Swift en hierdoor manual memory management verwijdert.
4. Oude deprecated API's omgezet naar moderne API's die op dit moment ondersteund worden door iOS.

Deze punten zullen voor het bedrijf geld en tijd besparen doordat taken die normaal gesproken in twee projecten moesten worden uitgevoerd, nu alleen in het framework hoeven te worden uitgevoerd. Hiernaast zullen de iPhone en iPad versie altijd een consistente code base hebben en hoeft hier geen tijd meer in worden gestoken.
Door de gemoderniseerde code base met Swift en moderne API's als doel kan het framework nieuwe iOS features gemakkelijker en efficiënter ondersteunen.
Het resultaat omvat vooral ook de mogelijkheid om zowel met een groter team als gelijktijdig aan beide apps te werken. Als laatste kan de code eenvoudig hergebruikt worden in andere apps.

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

Software ontwikkelmethode: Nog niet besloten
Team waarin opdracht wordt uitgevoerd: De opdracht voer ik alleen uit, met begeleiding van de mentor & eventueel hulp van collega ontwikkelaars.

Taak	**Dagen**	**Methodieken**
Plan van aanpak schrijven	4	
Vaststellen Requirements	5	Moscow
Ontwerpen	11	UML
Bouwen	27	Swift, Objective-C
Testen	12	Unit testing, logisch en fysieke testgevallen
Documentatie schrijven	5	
Implementatie	6	
Afstudeerverslag	15	

6. Op te leveren (tussen)producten

Plan van aanpak
Requirements rapport
Systeem Design Diagram (klassendiagram)
WeekCal Framework
Testplan & rapportage
Week Calendar app werkend met het WeekCal framework op iPhone en iPad
Documentatie WeekCal Framework

7. Te demonstreren competenties en wijze waarop

1.4 Uitvoeren analyse door definitie van requirements (Niveau 3)
Er worden requirements opgesteld om te analyseren waaraan het WeekCal framework moet voldoen. Deze worden deels opgeleverd door de opdrachtgever en deels zelf gespecificeerd uit interviews en eigen inbreng. Vervolgens moeten de requirements worden geprioriteerd met de MOSCOW methode, ik voer de prioritering zelf uit met als bron interviews met stakeholders.

3.2 Ontwerpen Systeemdeel (Niveau 4)
Het WeekCal framework moet worden ontworpen, twee code bases moeten een worden en kleine verschillen tussen iPhone en iPad functionaliteit moeten goed kunnen worden toegepast. Hiervoor is een ontwerp dat mogelijkheid geeft tot sterke uitbreidbaarheid nodig. Er moeten design patterns worden gebruikt die het ontwerp deze uitbreidbaarheid geeft. Fout gekozen design patterns zullen

grote impact hebben op het project; er moet dus veel aandacht worden besteed aan het kiezen van de juiste design patterns en hoe deze zullen gebruikt worden.

3.3 Bouwen Applicatie (Niveau 4)
Het WeekCal framework moet worden gebouwd, dit bestaat uit het implementeren van het ontwerp, Objective-C code omzetten in Swift code en verouderde code moderniseren.

3.5 Uitvoeren en rapporteren over het testproces (Niveau 3)
Het WeekCal framework moet volledig tastbaar zijn, hiervoor wordt er zelfstandig een test proces uitgevoerd en gerapporteerd. Door de grote hoeveelheid gebruikers in productie moet het proces correct en heel precies worden uitgevoerd

11. Schriftelijke rapportage bedrijfsmentor

THE HAGUE UNIVERSITY OF APPLIED SCIENCES

Faculty of IT & Design

Delft
The Hague
Zoetermeer

Evaluation form for graduating
To be filled out by commissioning institution and/or host institution's work-position coach(es)

Student: Guus Mulder

Period: 2016-1.2 (start uiterlijk 9 mei 2016)

Enterprise or. Institution: WeekCal

Work-position coach: Saurabh Garg

City/town: Wateringen

Date (d/m/y): 22/09/2016

1. Did the student soon and actively settle into the enterprise/institution and into executing the assignment set?

Yes, Guus was able to understand the project and the codebase fairly quickly and easily.

2. What is your judgement of the student's communicative skills (in her/his cooperation with colleagues, in contacts with clients/customers, in oral presentations, written reports)?

Very good. Guus was vocal in standups as well as communicating his progress. He also often shared his ideas on how things could be done better.

3. How did the student function in the execution of her/his assignment?

Regarding:	**Please encircle:**

- **responsibility** good / sufficient / moderate / insufficient
- **self-reliance/self-support** good / sufficient / moderate / insufficient
- **systematic/methodical work approach** good / sufficient / moderate / insufficient
- **creativity** good / sufficient / moderate / insufficient
- **productivity** good / sufficient / moderate / insufficient
- **cooperation with colleagues** good / sufficient / moderate / insufficient
- **creating on the job support** good / sufficient / moderate / insufficient
- **anticipating and dealing with company culture** good / sufficient / moderate / insufficient
- **observing/respecting the specific company context** good / sufficient / moderate / insufficient
- **getting necessary changes going** good / sufficient / moderate / insufficient

4. What is your judgement of the student's knowledge, know-how and skills in relation to what you expect of a near-graduate?

Guus has a wonderful grasp on the principles of software development, and especially development for iOS. In a lot of instances, I found his knowledge and experience to be better than that of a recent graduate.

5. What is your judgement of the quality of the (intermediary/semi-finished) products delivered?

Good. The quality is on par with what someone fresh out of school would produce.

6. Are you satisfied with the (final) product delivered?

- **To what extent did you get what was agreed on?**
  Fully. Since Guus was very involved in communicating about what he was working on, the results match the expectations.

- **To what extent does the (end) product meet your expectations?**
  Good. It helps in deciding the next steps.

- **How about its usability and sustained maintainability?**
  The usability is in all our products since the goal was to create a framework of our core functionality that could be just dropped into our applications. It will also be easier to maintain as a result.

- **What will happen with the (final) product as delivered?**
  The completed product will be evaluated and beta tested internally before it is adapted for public consumption.

- **Can you immediately put the product into use?**
  Yes.

7. Are there any aspects whatsoever important to you and that have not been addressed yet?

No.

8. Would you be willing to again make a 'graduation' position available?
(Please, explain)

Definitely. It was a good experience and I shall be willing to mentor another student so that they could become a wonderful software engineer.