

# Materiaali

Xcode on Applen tekemä IDE (Integrated Development Environment), jolla luodaan iOS, OS X, tvOS ja watchOS ohjelmia. Ensisijainen tiedonlähde on [Applen iOS kirjasto](#), josta löytyy vastauksia melkein kaikkiin kysymyksiin mitä voi keksiä, tosin kirjasto saattaa myös herättää kysymyksiä. Applen maailmassa eletään ikuisessa silmukassa (Infinite Loop, Cupertino, CA 95014, USA), joten dokumentaatio muuttuu melko vauhdikkaasti. Täytyy vaan pysyä “loopissa”.

Materiaalia löytyy iOS opetteluun melkoisissa määrissä netistä. Tähän materiaaliin pyritään kerätä parhaimmat mahdolliset lähteet. Jos, törmäät aivan loistavaan tai muuhun hyvään materiaaliin, niin jaa se mielellään kurssin ohjaajalle.

## Xcode

Tutustutaan aivan aluksi Xcoden eri osiin käymällä läpi [Xcode overview](#). Huomaa, että tiettyjen ikonien ja käyttöliittymän ulkoasu on saattanut muuttua hieman materiaaliin verrattuna, ikonien ja osien sijainti uusissa versioissa Xcodessa on pääsääntöisesti kuitenkin sama.

## Interface Builder

Käyttöliittymän rakentaminen voi tapahtua kolmella eri menetelmällä; **Storyboard**, **XIB** tai suoraan koodattu. [Interface Builder](#) on työkalu Xcodessa, jolla rakennetaan käyttöliittymiä muokkaamalla [Storyboard](#) tai **XIB** tiedostoja (ovat XML muodossa ennen kääntöä). Käyttöliittymäelementtejä (UIView, UIButton jne.) lisätään yksinkertaisella drag-n-drop menetelmällä ([adding an object to your interface](#)). Storyboard ja **XIB** eroaa siinä, että Storyboard sisältää monta näkymää ([Scene](#)) ja näkymien siirtymiä ([Segue](#)), jonka avulla pystyy luomaan navigointipolkuja, siirtymällä näkymästä toiseen. **XIB** on taas yhden näkymän tai käyttöliittymäelementin tiedosto (tosin, yhdessä näkymässä voi tietenkin olla paljonkin käyttöliittymäelementtejä). Vaihtoehtoisesti on tietenkin mahdollista luoda näkymiä kuvaamalla ne täysin koodilla.

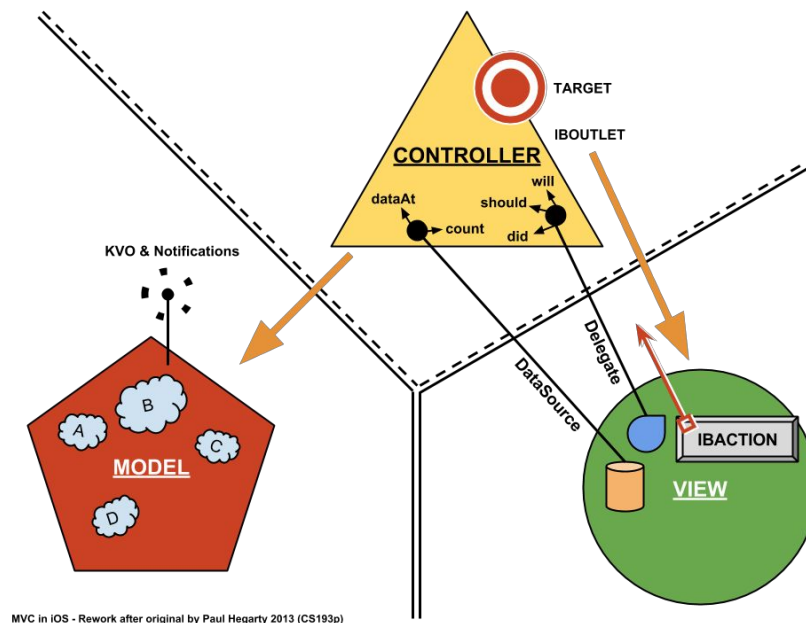
Mitä menetelmää pitäisi sitten käyttää? Tämä valinta on täysin projektin tarpeista kiinni. Jokaisella vaihtoehdolla on [hyviä ja huonoja puolia](#).

Jollain menetelmällä pitää kuitenkin kytkeä käyttöliittymäelementit koodiin. Tämä tapahtuu luomalla ns. [Action Connection](#). Päinvastainen kytkentä, koodista käyttöliittymäelementtiin luodaan ns. [Outlet Connection](#). Luotuja yhteyksiä voidaan muuttaa [Connection Panel](#) palikan avulla.

# Simulaattori

Tietyn Xcode-version mukana tulee aina simulaattori sen hetkiselle iOS versiolle. Esimerkiksi Xcode 7.1.1 sisältää kaikki laitteet, jolla on käyttöjärjestelmänä iOS 9.1. Halutun laitteen simulaattori valitaan työkalurivistä. Jos simulaattori alkaa käyttäytyä huonosti voit resetoita sen täysin menemällä (simulaattori käynnissä ja valittuna) Simulator > Reset Content and Settings..

## Model View Controller



Kuva 1. MVC

Smalltalkin ajoilta peräisin oleva arkkitehtuurityyli, [Model View Controller](#) (MVC) näkyy monessa rakenteessa.

## View ja Controller

Jokaisella näkymällä on ns. ViewController, kuvassa 1 näkyvä Controller onkin oikeasti useimmiten ViewController. View on käyttöliittymää kuvaavat lähteet. Tapahtumat käyttöliittymästä ViewControlleriin seuraa ns. [Target-Action](#) design patternia, jossa määritellään tapahtumalle (action) vastaanottajan (target). Toinen useassa käytössä oleva design pattern, joka myös näkyy kuvassa 1 on ns. [Delegation \(& Data Source\)](#).

## Model

Jäljelle kuvassa 1 jääkin model, eli malli missä muodossa tietoa säilötään ja tarjotaan. Tiedon muutoksista on syytä pitää malli hyvin pitkälti tietämättömänä mistään muusta maailmasta. Tarjotaan tietoa vain niille ViewController-olioille, jotka ovat kuuntelemassa tai tarkkailemassa tiettyjä muutoksia.

## iOS arkkitehtuuri

Kokoelma teknologioita erityisesti iOSia varten korkeimmalla tasolla nimetään [Cocoa Touch](#). Täytyy muistaa, että iOS käyttää myös teknologioita [Cocoa](#)-tasolta (jos tietty kehys on vain OS X käyttöön se on tavallisesti ilmaistu). Minkä tilojen välillä applikaatiot liikkuu? Mitä iOS (kehyksiä, kirjastoja yms.) tarjoaa applikaation tekemiseen?

- [App Programming Guide for iOS](#) ([iOS Technology Overview](#))

UIKit on esimerkiksi kehys korkeimmalla Cocoa Touch tasolla, jolla käsitellään näytön kosketus-tapahtumia. Toisin sanoen, Cocoa Touch on iOS applikaatioiden ympäristö.

## Objective-C

Ohjelmointikieli mitä käytämme tällä kurssilla on Objective-C. Apple on kehittynyt myös modernin kielen applikaatioiden rakentamiseen nimeltä Swift. Vaikka uuden Swift kielen käyttöaste kasvaa, Apple ylläpitää ja kehittää yhä Objective-C:tä (julkaisee myös tuoreita esimerkkejä Objective-C:llä toteutettuna).

- [Programming with Objective-C](#) ([Concepts in Objective-C Programming](#))

## Foundation

Tämä kehys on koko ympäristön pohja. Se sisältää mm. NSObject, joka on kaikkien? olioiden isä. Kannattaa tutustua ainakin pinnalta, mitä kaikkea [Foundation](#) tarjoaa.

## Lisää ViewController rooleista

Jos tarkastaa hieman kuvaa 1, niin voi kuvitella, että ViewController, erityisesti jos vain yksi koko applikaatiossa, saa aivan liian paljon vastuuta tehtävistä. Jossain kohtaa on pakko miettiä miten näkymät oikein jakautuu. Ehkä pitää siirtää jotain ihan omaan näkymään. Vastuun jakamiseksi on määritetty kaksi eri roolia, jota ViewController voi esittää. Yksi näistä on *Container View Controller*, joka on tarkoitettu sisältämään muita ViewControllereita ja

tarjoamaan navigoinnin näiden välillä. Toinen on “tavallinen” ns. Content View Controller, joka sisältää näkymät. Tarkemmin asiasta pystyy lukemaan mm. [täältä](#).

Tavalliset Content View Controllerit:

- UITableViewController
- UICollectionViewController
- UIViewController

Tavalliset Container View Controllerit:

- UINavigationController
- UITabBarController
- UISplitViewController

## Tarkempaa Model touhua

Tiedon säilyttämiseen ja mallintamiseen on hyvä valita sopiva menetelmä. Jos ei valinnut jotain järjestelmällistä mallia, niin ennemmin tai myöhemmin tulee kohtaan tilanne, jossa jatkokehitys on hankalaa. Esitetään lyhyesti kaksi eri hyvin suosittua ja järkevää menetelmää.

### Core Data

Jos on tarve käsitellä tietoa tehokkaasti, kannattaa tutustua Core Data kehykseen. Hienojen abstraktien rakenteiden alla piileksii SQLite tietokanta, jota ei kannata ilman parempaa tietoa suoraan sorkkia. Core Data tarjoaa mm. olio-verkon (eng. object graph) ja pysyvää tallennusta. Core Data oliot antaa monta hyvää, ne voi mm. tallentua automaattisesti, iCloud valmiuden.

### UIDocument

Dokumentti pohjaisille applikaatioille on mahdollista mallintaa olioita, joilla mm. automaattinen tallennus. Jos tämän lisäksi haluaa tallentaa iCloudiin voi käyttää UIManagedDocument luokkaa. iCloud vaatii kuitenkin lisävarusteita (mm. Developer Lisenssin).

### User Defaults

Pienen määrän dataa (asetuksia yms.) on mahdollista myös tallentaa avain-arvo säiliöön, tunnettu nimellä [NSUserDefaults](#).

## Verkkoyhteyden monitorointi

Jos applikaatio käyttää verkkoyhteyttä on syytä tarkkailla laitteen yhteys verkkoon. Voi tällöin mm. indikoida jollain elementillä tai ilmaisulla jos yhteys (joka on välttämätön toimivuuteen) katkeaa. Apple tarjoaa [Reachability](#) luokan tähän tarkoitukseen. Tallenna vaan .h ja .m tiedostot

omaan projektiin ja aloita käyttö (jos applikaatio menee jakeluun niin ota LICENSE.txt tiedosto mukaan).

## Testaus

Applikaation testaamiseen on käytettävissä melko laaja testikehys nimeltä XCTest, jolla voi mm. mitata toimintakykyä (performance measurement), testata käyttöliittymää (UI Tests) ja applikaation komponentteja ns. yksikkötesteillä (Unit Tests). Kun luo uuden projektin, voi valita haluaako sisällyttää valmiiksi luodut pohjat yksikkötesteille ja/tai käyttöliittymätesteille. Lue lisää [Testing with Xcode](#) dokumentista.

## Fox

Jos on kiinnostunut hieman enemmän automatisoidusta komponentti testauksesta, niin ominaisuuksiin-perustuva testaus (property-based testing) on mahdollista näppärällä kirjastolla nimeltään [Fox](#).

## Continuous Integration

Kaikki opiskelijat, joilla on [GitHub Education-plan](#) (jos ei vielä ole, mitä odotat? Saat mm. heti käyttöön 5 yksityistä repoa) voivat myös lisätä itsensä [Travis-CI Education-plan](#) käyttäjäksi. Kytket helposti Travis-CI kautta käyttöön minkä tahansa oman GitHub repoon, jonka buildaus käynnistyy automaattisesti joka pushausta kohti. Jos lisäät seuraavat pari-kolme YAML riviä repoon, niin Travis-CI ymmärtää mitä oikein pitäisi tehdä:

```
language: objective-c
script:
  - xctool -project JOSKANSIO/MunProjekti.xcodeproj -scheme MunProjekti -sdk iphonesimulator
    -destination "OS=9.1,name=iPad 2"
  - xctool test -project JOSKANSIO/MunProjekti.xcodeproj -scheme MunProjektiTests -sdk
    iphonesimulator -destination "OS=9.1,name=iPad 2"
```

Huomaa, että saat muokata polun, schema-nimen, simulaattorin ja mille laitteelle buildaus olisi rakennettava. Ensimmäinen xctool kutsu on itse applikaation rakentaminen ja toinen xctool kutsu on testien suoritus. Varmista [täältä](#), että valitsemasi simulaattori ja iOS SDK on tarjolla.

## Lyhyesti version ja buildin numeroinnista

Vaikka pienessä projektissa työskentelee, voi olla järkevää omaksua ns. semanttisen menetelmän [versiointiin](#). Tämän menetelmän mukaan on version tunniste jaettu kolmeen osaan (MAJOR.MINOR.PATCH). Build-numeroa kasvatetaan tavallisesti kun applikaatiolle tehdään päivityksiä sisäisiä pikku muutoksia/korjauksia/lisäyksiä, jotka eivät vaikuta ulkomaailmaan, eli se on kuvaa useimmiten karkeasti commit-määrää.

# Koodin staattinen analyysi

TBA

## Hyödyllisiä lähteitä

Kokonainen kurssi (noin 20 tuntia), suosittelen ainakin kolme tai viisi ensimmäistä.

[Developing iOS 7 Apps by Paul Hegarty - Stanford University](#)

[NSHipster](#)

[objc.io](#)