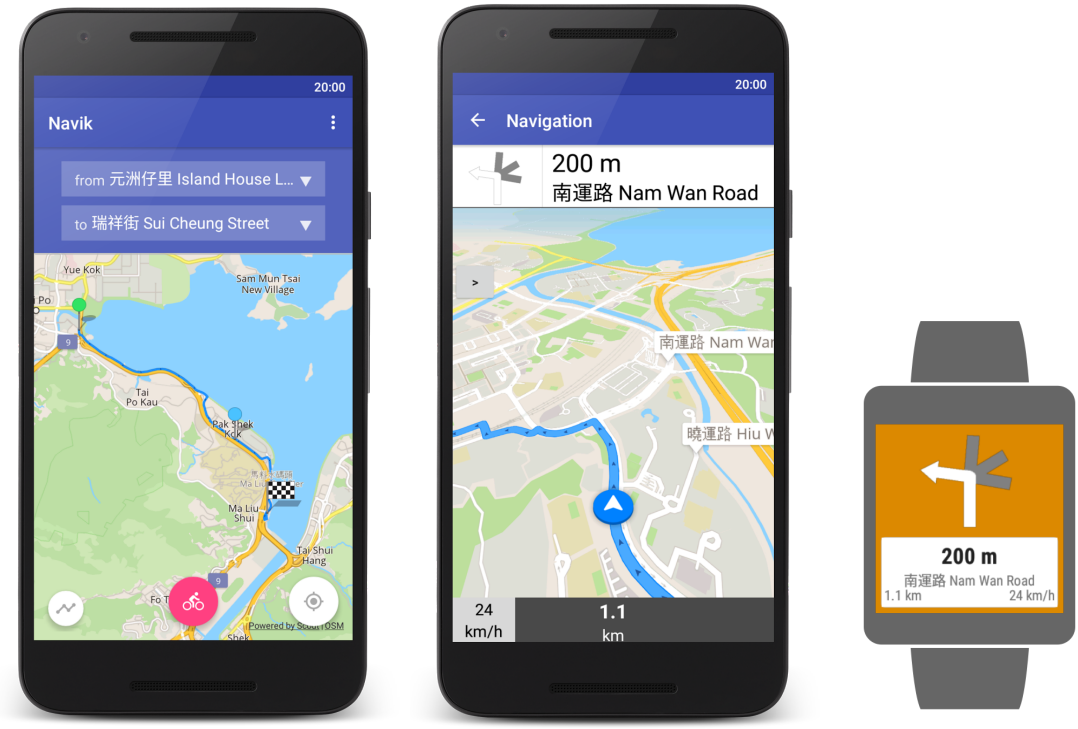


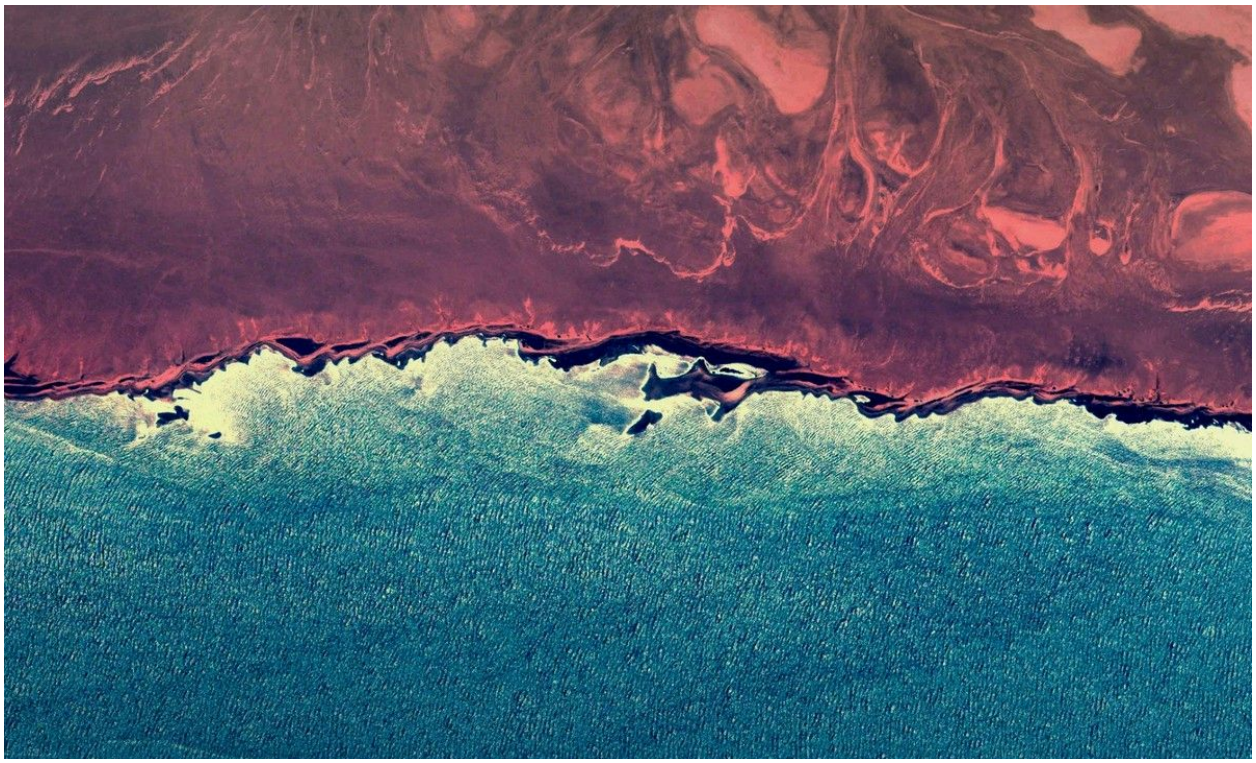
Programming an intelligent watch

Navik - Cycling navigation, with smartwatch compatibility

Final Report

CSIS0801 Final Year Project, Department of Computer Science, The University of Hong Kong

0 Table of Contents



1 Introduction

1.1 Abstract

Navik is a **cycling navigation** solution, with **route planning**, **route analysis** and **turn-by-turn navigation** ability.

Navik aims at providing **offline**, **low cost** and **cycling specific** alternative to existing solutions.

Just a glance at your wrist. Enjoy an **uninterrupted cycling experience** with the **extended display** and **notification** on your fashionable **smartwatch**.

As a cycling enthusiast, I proudly introduce **Navik** to you.

1.2 Team

Student: Lam Ka Fun Gavin

Supervisor: Prof Lau Francis

Project site: https://i.cs.hku.hk/fyp/2015/fyp15024

Contact e-mail: me@gavin.hk

2 Overview

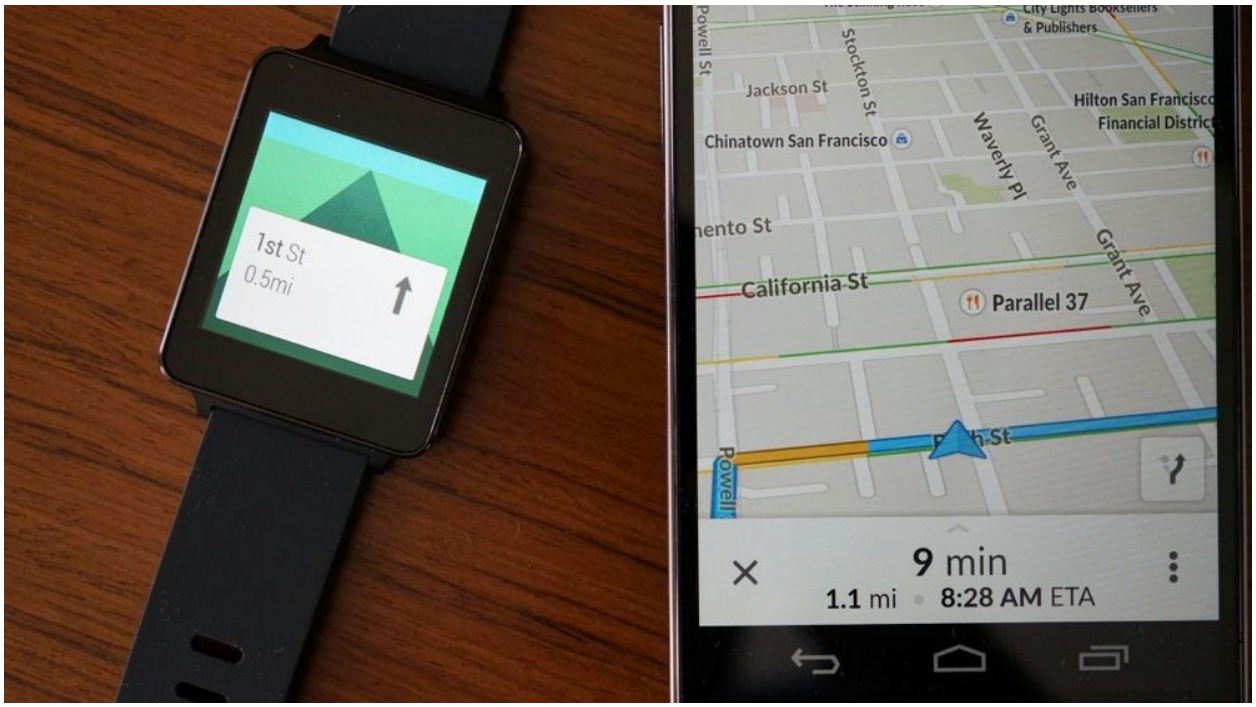
2.1 Project Background

There are various navigation solutions available on the market. **However, for cycling? Limited.**

If you look towards the **ordinary navigation solutions**, it **does not suit very well**. It is likely that you would end up finding yourself riding on a highway or into a cross harbour tunnel. Moreover, such solutions cannot satisfy your **unique requirements for cycling**. In addition, what if you have no data connection when you are riding in the middle of nowhere?

If you look towards **bike specific navigation solutions**, make sure you have a deep pocket. A bike computer with navigation functionality would probably **costs more than your bike**. Besides, why bother purchasing a specific navigation device when we are living in the 'smartphone era'?

2.2 Existing Solutions Analysis



2.2.1 Google Maps with Navigation

Google Maps is a cross platform **online application** with comprehensive maps throughout the globe. Voice-guided navigation is available for driving, public transit and walking. Live traffic condition, street view and point of interests information are also available.

Advantages

- **Free** of charge
- Compatible with **smartwatches**
- **Route construction** with **intermediate destinations**
- Provide **maps** and **turn-by-turn navigation** display

Disadvantages

- Cannot load **custom routes**
- No **offline navigation support** and limited **offline map data**
- Not support **round trip navigation**, which is important for cycling
- No **cycling specific navigation** and related functionalities available

2.2.2 Garmin Bike Computer with Navigation

Garmin produces **bike computers** of different levels and prices. Model with navigation functionality ranges from **USD$249.99 to USD$599.99**. User can connect the device for data transmission with computers via cable or the companion smartphone app via Bluetooth.

Advantages

- **Feature rich** bike computer
  - Compatible with wireless bike sensors (not included), such as speed sensor, cadence sensor, heart rate monitor and power meter
- Provide **cycling specific route construction** with **round trip** and **intermediate destinations** support
- Support **external routes import**, such as GPX files
- Provide **maps** and **turn-by-turn** display
- Provide **route analysis**, such as distance and elevation data
- **Offline map data** by OpenStreetMap

Disadvantages

- **Expensive** for casual and amateur riders
- Require **manual update** for system and map data
- **Closed** and **proprietary** system, thus the device is very specific and cannot have other use

2.2.3 HammerHead

HammerHead is an innovative **navigation device** mounted on your bike. The device will notify users with **light signals** when you need a turn.

Users have to connect the device with their own **navigation app** running on your smartphone. They offer the physical device for **USD$104.99** (with mount) and the companion smartphone app for free.

Advantages

- Provide **cycling specific navigation** with **intermediate destinations**
- Support **external routes import**, such as GPX files
- Cross platform **route sharing** via iMessage, Email, Twitter, Facebook and Whatsapp

Disadvantages

- **Expensive** for the physical device with limited usability
- **Easy to miss** the turning light signals
- There are **limited information** provided on the display of the device
    - For example, map and route display would be useful
- Riders typically mounts cyclometer on their bike, having an **extra device** mounted would not be an elegant solution
    - Messy mounts, extra weights and air resistant are not preferred
- Using **online map and navigation** service
- Not support **round trip navigation**, which is important for cycling

2.3 Objectives

2.3.1 Route Planning

Allow user to **plan a route** by providing **starting point**, **destination** and **waypoints**.

Equip with **cycling specific auxiliary route planning features** as well.

2.3.2 Route Analysis

Allow user to **have an insight** into the planned route.

2.3.3 Turn-by-turn Navigation

Give **turn-by-turn navigation** on planned route.

2.3.4 Smartwatch Capability

Allow user to look at **general instruction** and **get notified** during navigation with **smartwatch** connected.

2.4 Requirements

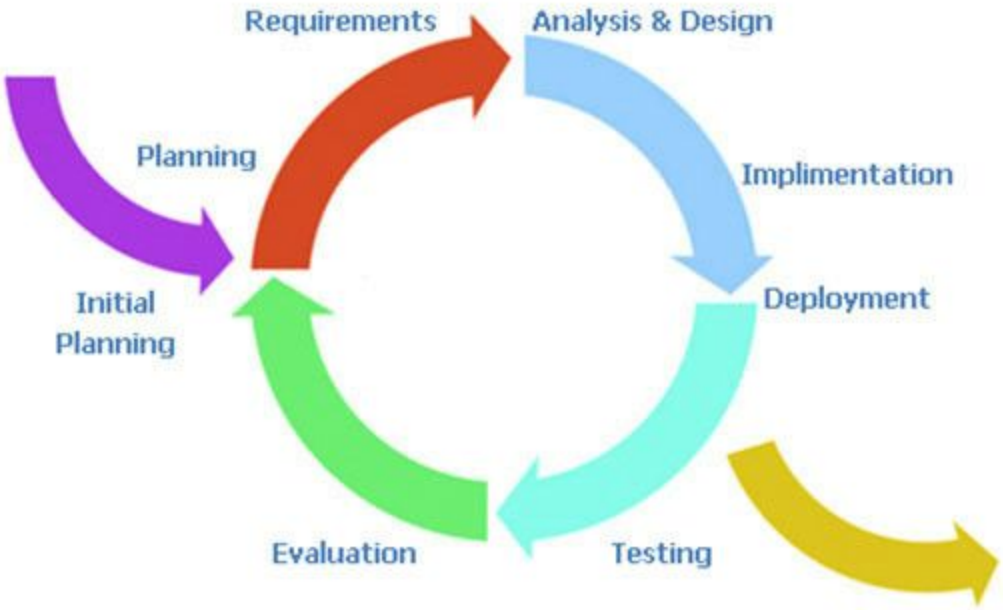
An **Android application** running on smartphone of latest Android version

- Provide **offline** map display
- **Construct route** with **round trip** and **intermediate destination**
- Provide **external routes import** via **GPX files**
- Provide **route analysis**, such as **distance** and **elevation data**
- Provide **turn-by-turn navigation**
- Provide **real-time information**, such as speed and distance remaining

An **Android Wear** companion application

- **Turn-by-turn navigation** with **intuitive visual instruction** and **vibration** notification
- Provide **real-time information**, such as speed and distance remaining

3 Methodology

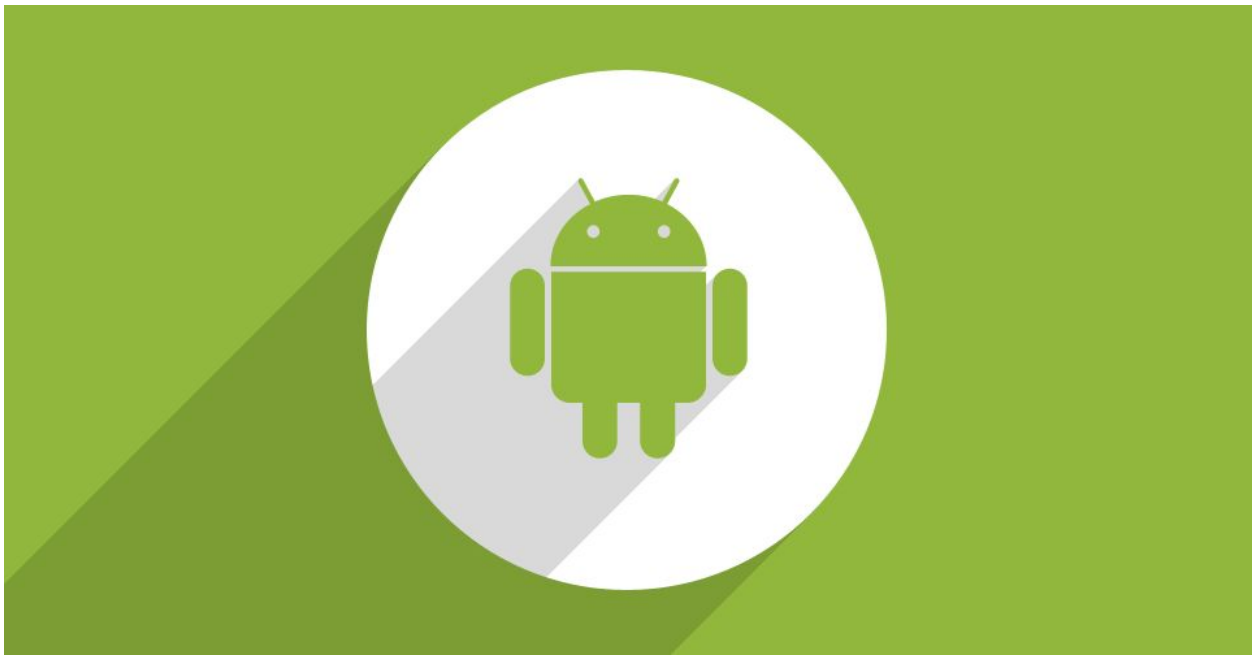


3.1 Software Development Process

Incremental model will be applied.

The whole project is divided into several **incremental builds**. **Working implements** will be delivered after each **milestones**. Each release adds functions to the previous release.

Thus enabling **faster delivery** of working release and **more flexible design**. **Risks** are **easier to managed** as risky components are addressed individually.



3.2 Platform

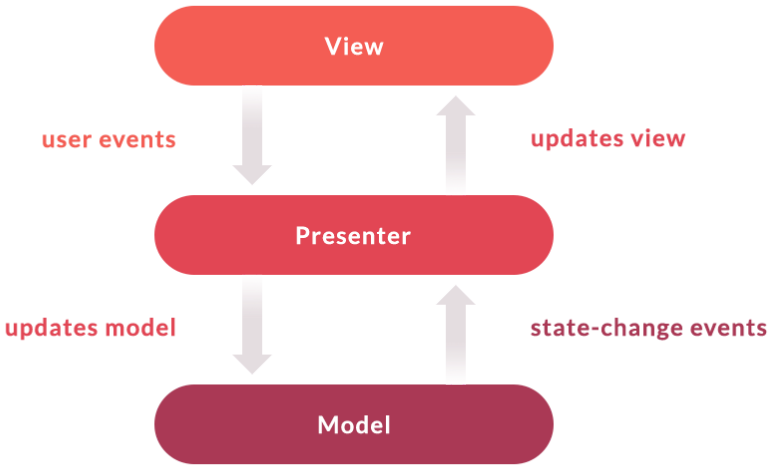
- **Java 1.7** on **Android OS**
- Build for **Android 6.0 Marshmallow**
- **Backward compatible** from **Android 4.0 Ice Cream Sandwich** to **Android 5.1 Lollipop**

3.3 Project Dependencies

- **Gradle 1.5** for dependency management
- **Guava**, **Apache Commons Lang3** and **Lombok** for enriching Java functionality
- **Dagger** for **dependency injection**
- **ButterKnife** for **view and resources injection**
- **JDeffered** for **promise pattern**
- **Android support library**, **Android Appcompat library** and **Android design library**, **Android Cardview** for ensuring backward compatibility
- **Google Play Services Wearable** for communication with Android Wear devices
- **Skobbler SDK 2.5.1** powered by **OpenStreetMap** for the map rendering, map data, route construction and navigation
- **Google Elevation API** for elevation information query
- **MPAndroidChart** for **chart rendering**
- **NoNonsense-FilePicker** for file selection

3.4 System Architecture

3.4.1 Model View Presenter (MVP)



Model View Presenter is an **architectural pattern** for clear separation between presentation and business logic. It **eases maintenance** and make the code more manageable.

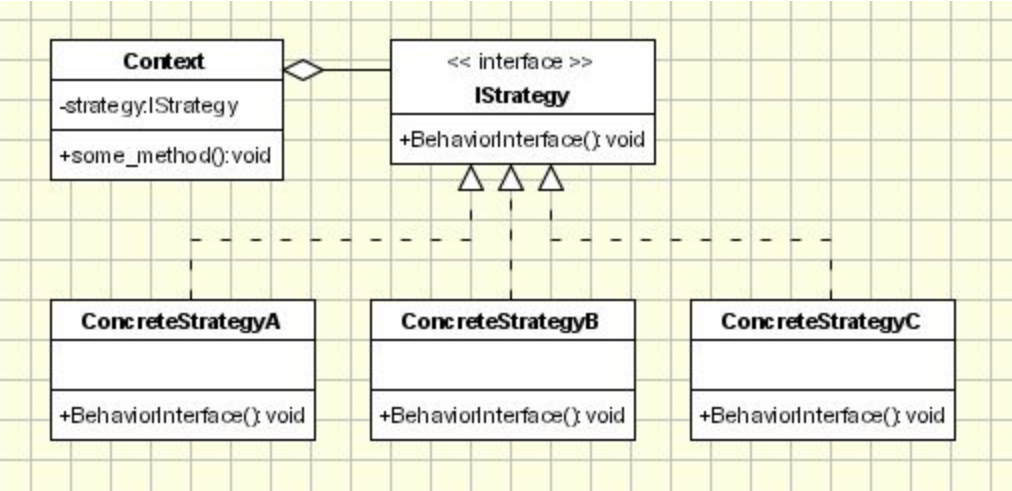
Model represents pure domain model which serves as the **data layer** and **business layer**.

View contains the **UI components** and related event handling logic.

Presenter handles the **communication between models and views**.

For this project, models are the **data models** and **business logics**. Views are the **Android views**, **fragments** and **activities**.

3.4.2 Strategy Pattern

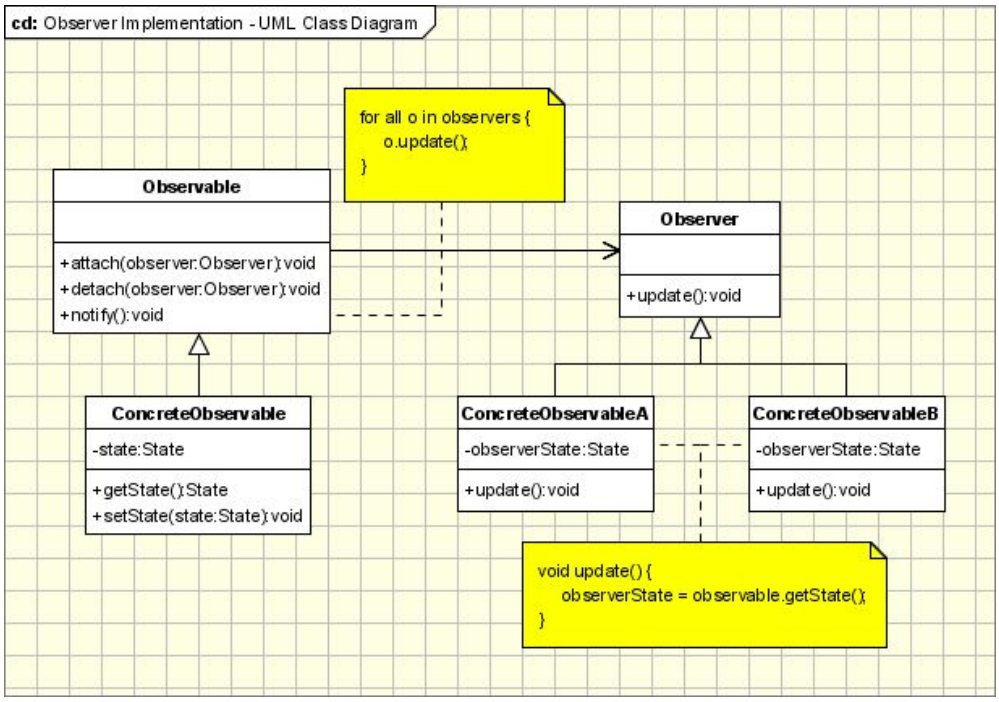


Strategy pattern splits the behaviour from the interface. It **isolates the algorithms in separate classes** in order to have the ability to select different algorithms at runtime. The caller only have knowledge to the **strategy interface** instead of how the concrete strategy works.

This project heavily makes use of this pattern to ensure **forward compatibility** and **ease maintenance**. It separates the caller concern from the underlying SDK and libraries implementation.

For example, **elevation module** makes use of the strategy pattern to abstract the concrete implementation of elevation data service making use of Google Elevation API. If I decide to switch to other services or algorithm for getting elevation data, I can just add another concrete strategy implementation.

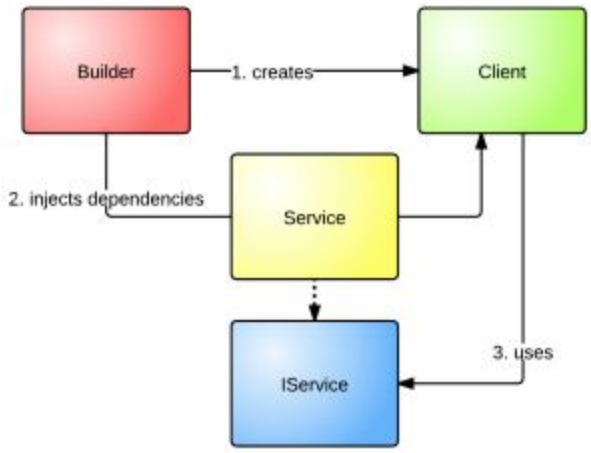
3.4.3 Observer Pattern



Observer pattern allows the **change in state in an object to be reflected in another object** without tightly coupling them. It **decouples** objects interacting with each other and reduce the dependencies.

In the project, observer pattern are applied in **user interface event handling** and **results notification of asynchronous works**. For example, **location module** relies on this pattern to broadcast the **live location change** to the subscribers. Thus allowing other location aware modules to react according to the location change.

3.4.4 Dependency Injection

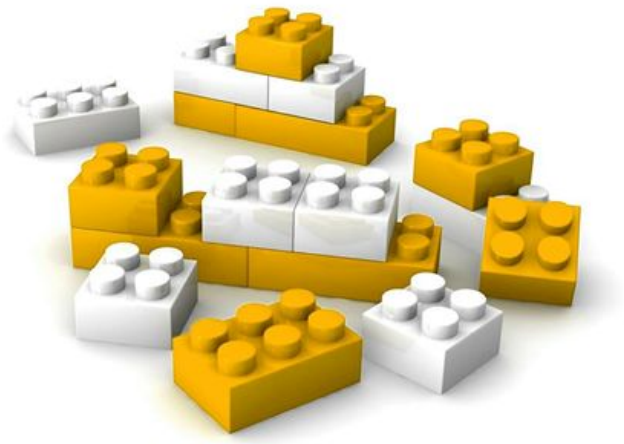


Dependency injection allows dependencies of objects to be injected and managed in a clear manner. It facilitates **inversion of control** and **decoupled architecture**. It also makes it easy to create **reusable** and **interchangeable** modules.

Dependency injection is done with **Dagger** in this project. Dependencies of fragments and presenters on other modules are injected according to the **dependency graphs** defined.

For example, the **location module** would be injected into the **location selection fragment** at runtime. Thus the fragment would be **decoupled** from the creation logic of the location module and **increase interchangeability**.

4 Design



The project is designed in **modular** and **multilayered** approach. The project composed of **separate components** that are designed with specific work. It allows greater **interchangeability**, **maintainability** and **readability**.

4.1 Contract Module

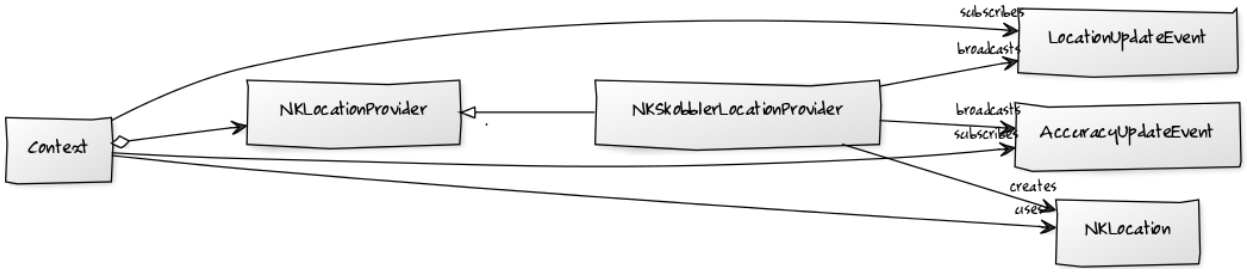
Contract module defines **container for constants** shared throughout the whole project. It facilitates future modifications and improve readability.

Name	Type	Description
UiContract	Class	Container for UI related constants. Including request and result code, fragment tags, etc..
WearContract	Class	Container for constants for communication between mobile application and wear companion application.

4.2 Core Module

Core module contains the **business layer** of the project. It is further subdivided into multiple modules of distinct responsibility.

4.2.1 Location Module



Location module contains **data model** for representing a location and location provider strategy for **live location update events**.

Name	Type	Description
NKLocation	Class	**Data model** for representing a location with coordinates.
NKLocationProvider	Abstract Class	Abstract class that defines the live location update strategy.
NKSkobblerLocationProvider	Class	**Concrete strategy** for live location update events implemented with Skobbler SDK.
LocationUpdateEvent	Class	Represents a location update event.
AccuracyUpdateEvent	Class	Represents a location accuracy update event.

4.2.2 Geocode Module

Geocode module contains **reverse geocode strategy** and planned to include geocode strategy in future works.

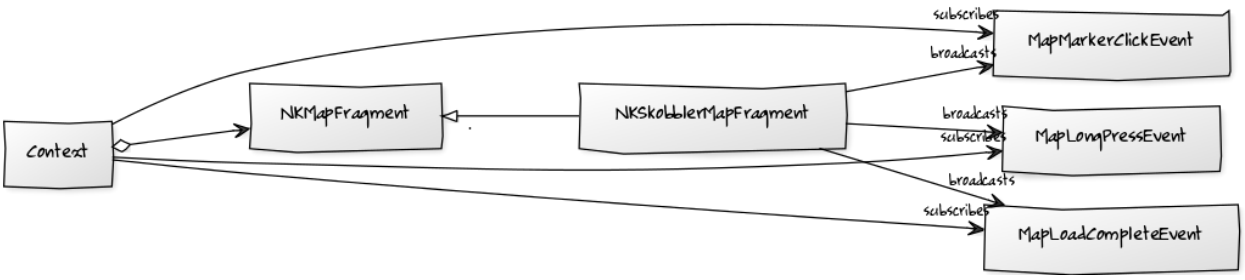
Name	Type	Description
NKReverseGeocoder	Interface	Interface that defines the reverse geocoding strategy.
NKSkobblerReverseGeocoder	Class	**Concrete strategy** for reverse geocoding implemented with Skobbler SDK.

4.2.3 Elevation Module

Elevation module contains **elevation resolving strategy**.

Name	Type	Description
NKElevationProvider	Abstract Class	Abstract class that defines the elevation resolving strategy.
NKGoogleElevationProvider	Class	**Concrete strategy** for elevation resolving implemented with Google Elevation API.

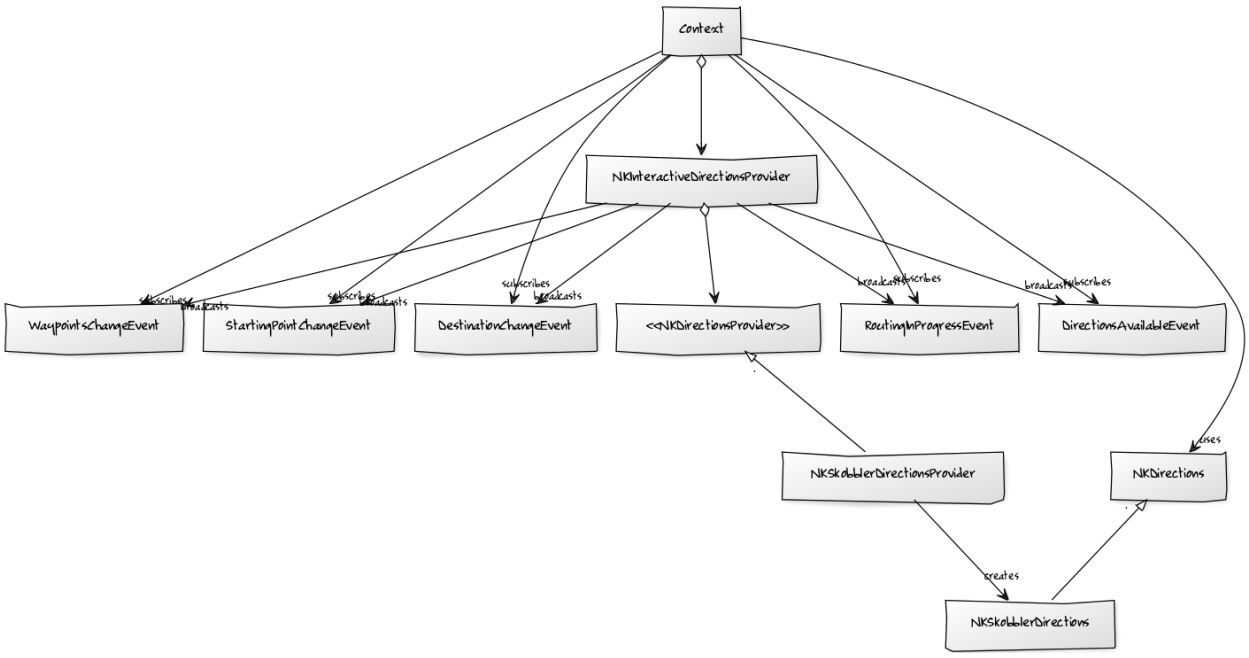
4.2.4 Map Module



Map module contains **map rendering and interaction strategy**.

Name	Type	Description
NKMapFragment	Abstract Class	Abstract class that defines the map rendering and interaction strategy. Defining functionalities such as subscribe to map events, move to current location, display markers, etc..
NKSkobblerMapFragment	Class	**Concrete strategy** for map rendering and interaction implemented with Skobbler SDK.
MapLoadCompleteEvent	Class	Represents a map load complete event.
MapLongPressEvent	Class	Represents a map long press event.
MapMarkerClickEvent	Class	Represents a map marker click event.

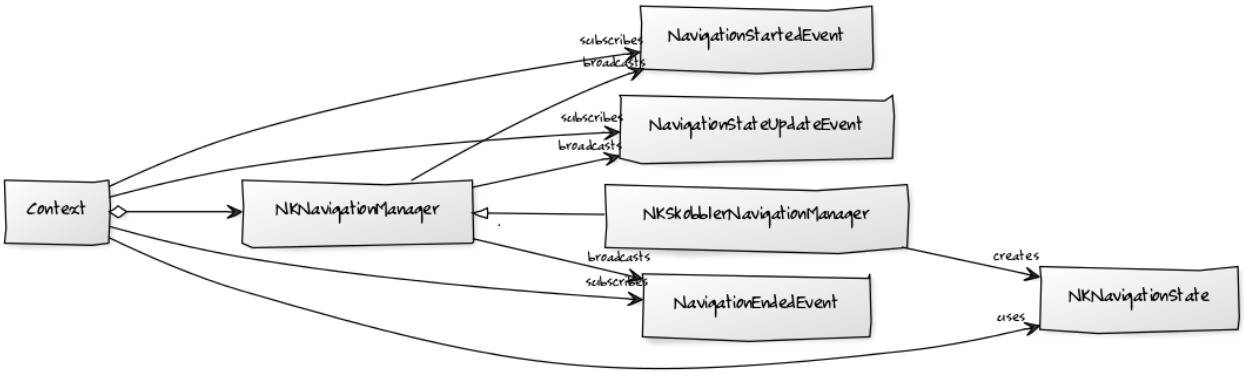
4.2.5 Directions Module



Directions module contains **data model** representing directions and **directions finding strategy**.

Name	Type	Description
NKDirections	Abstract Class	**Data model** for representing directions with starting point, destination, waypoints, all intermediate locations and other information.
NKSkobblerDirections	Class	Subclass of NKDirections with additional information specific to implementation using Skobbler SDK.
NKDirectionsProvider	Interface	Interface that defines directions finding strategy.
NKSkobblerDirectionsProvider	Class	**Concrete strategy** for directions finding implemented with Skobbler SDK.
NKInteractiveDirectionsProvider	Class	**Builder pattern** for directions finding with information provided part by part. For example, provide the starting point first, provide the destination later and change the starting point afterwards. Directions finding would be **delegated to the directions provider injected at runtime**.
StartingPointChangeEvent	Class	Represents a starting point change event.
DestinationChangeEvent	Class	Represents a destination change event.
WaypointsChangeEvent	Class	Represents a waypoints change event.
RoutingInProgressEvent	Class	Represents a routing in progress event.
DirectionsAvailableEvent	Class	Represents a directions available event.

4.2.6 Navigation Module



Navigation module contains **turn-by-turn navigation strategy**.

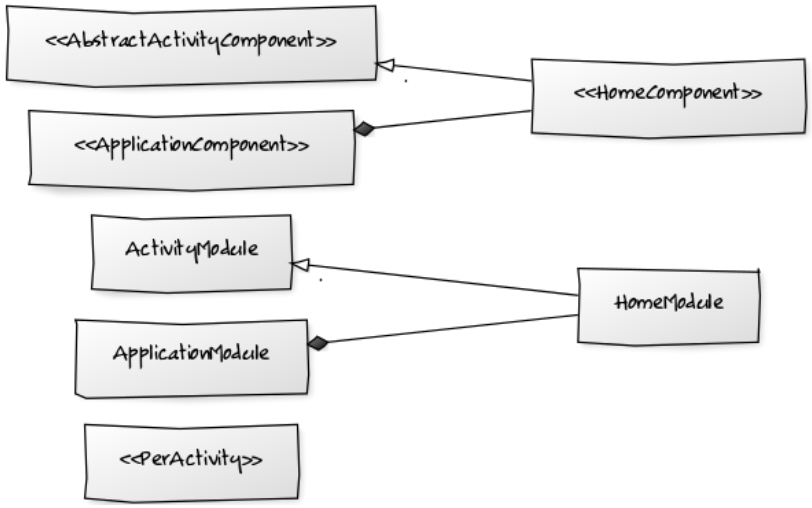
Name	Type	Description
NKNavigationState	Class	**Data model** representing a navigation state. Composes of data such as next street name, distance to next advice and current speed.
NKNavigationManager	Abstract Class	Abstract class that defines the turn-by-turn navigation strategy.
NKSkobblerNavigationManager	Class	**Concrete strategy** for turn-by-turn navigation implemented with Skobbler SDK.
NavigationStartedEvent	Class	Represents a navigation started event.
NavigationStateUpdateEvent	Class	Represents a navigation state update event.
NavigationEndedEvent	Class	Represents a navigation ended event.

4.2.7 Wear Module

Wear module is responsible for **communication with wear companion application**.

Name	Type	Description
NKWearManager	Class	Class that is responsible for **connecting to** and **communicate with** wear companion application.

4.3 Injection Module

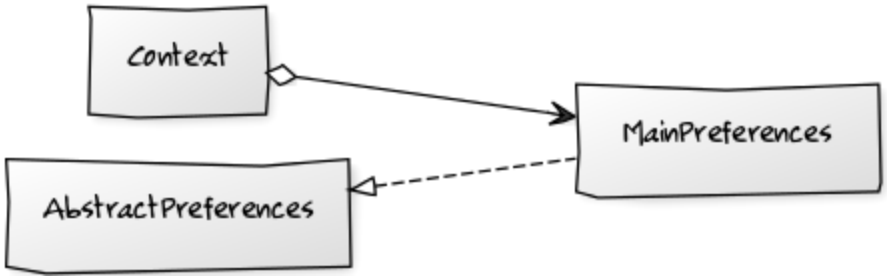


Injection module defines the **dependency graphs** for dependency injection at runtime. It is done with **Dagger** dependency injection library.

Component interfaces are the representation of a component in the dependency graph. **Module classes** are the representation of a module that contributes to the dependency graph.

Name	Type	Description
ApplicationComponent	Interface	Component representation of the application object in the dependency graph.
ApplicationModule	Class	Module representation of the application object in the dependency graph.
AbstractActivityComponent	Interface	Abstract component representation of the activity objects in the dependency graph.
ActivityModule	Class	Abstract module representation of the activity objects in the dependency graph.
HomeComponent	Interface	Component representation of the home activity object in the dependency graph.
HomeModule	Class	Module representation of the home activity object in the dependency graph.
PerActivity	Annotation	Annotation for marking dependency that differs per activity.

4.4 Preference Module

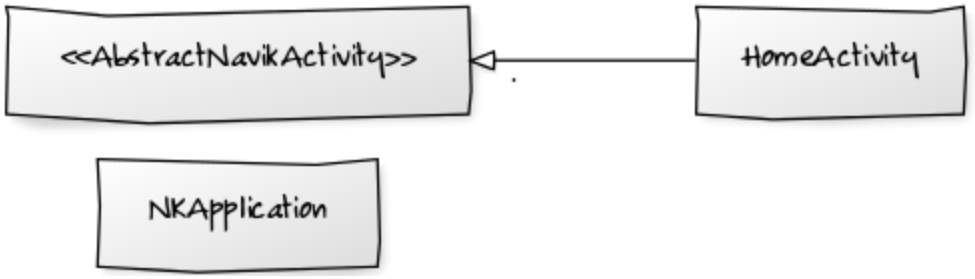


Preference module is responsible to deal with **shared preferences**, which is the **key value storage mechanism** provided by Android platform. It allows retrieval and update of simple persistent data.

Name	Type	Description
AbstractPreferences	Abstract Class	Abstract class that provide retrieval and update mechanism of simple data type for subclasses.
MainPreferences	Class	Provide retrieval and update service of named key value data.

4.5 User Interface Module

4.5.1 Application and Activity Module

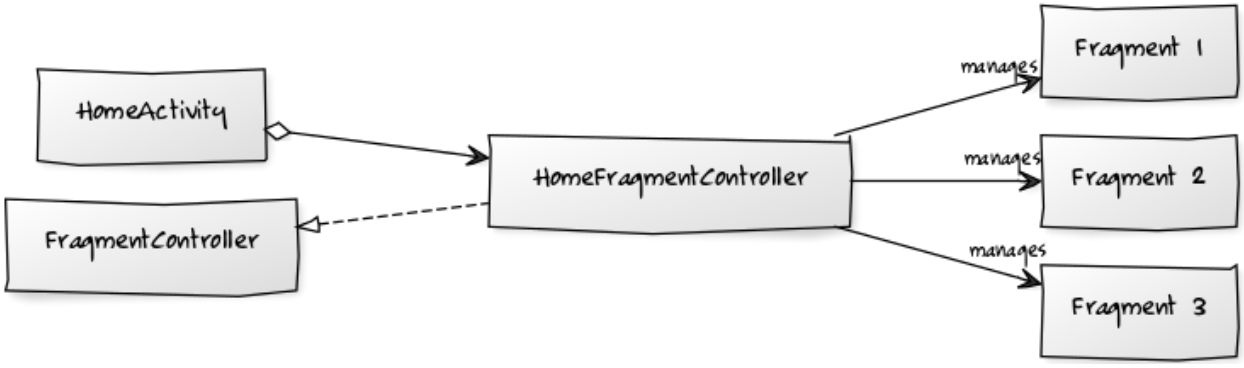


Application and **activities** are important parts of an Android application that deals with **lifecycle events** and **interactions with user**.

For this project, they plays an critical role in the **dependency injection pattern** also.

Name	Type	Description
NKApplication	Class	Entry point of the mobile application. **Build the dependency graph** for the whole project.
AbstractNavikActivity	Interface	Interface that defines the common behaviour of activity classes.
HomeActivity	Class	Entry activity of the mobile application that reacts to **Android activity lifecycle events** and **build the dependency graph** for fragments and presenters.

4.5.2 Fragment Controller Module

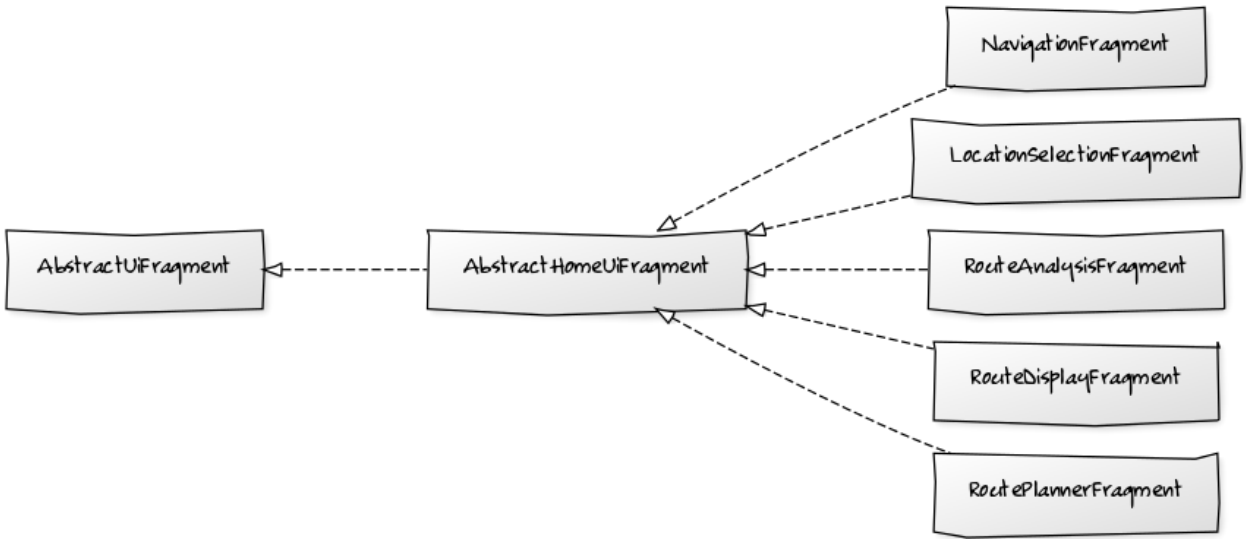


A **fragment controller** manages its member fragment.

It manages **transaction of fragments**, provides **request and result mechanism** for transaction of fragment and manages **shared UI components**.

Name	Type	Description
FragmentController	Abstract Class	Abstract class that provide basic feature for subclasses.
HomeFragmentController	Class	Manages fragments associated with home activity.

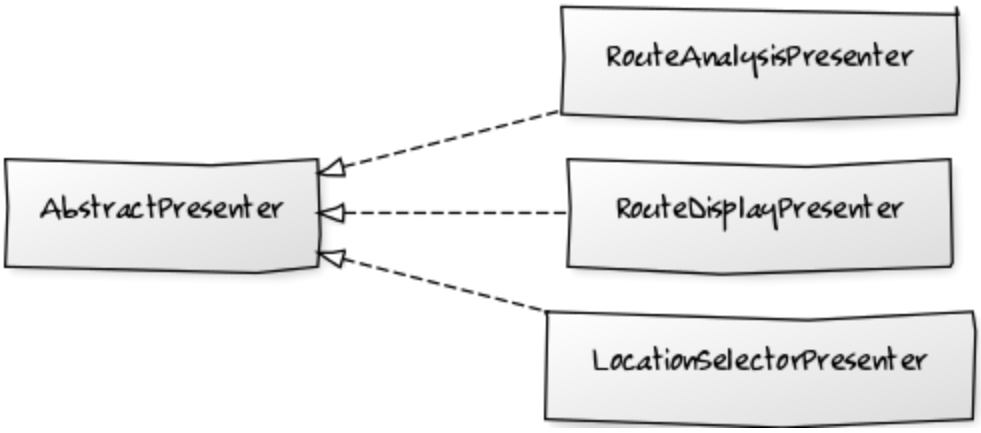
4.5.3 Fragment Module



Fragment module contains of **UI fragments** that represent a page in the mobile application. Fragments follow their lifecycle events and deals with user interaction.

Name	Type	Description
AbstractUiFragment	Abstract Class	Abstract class that define basic behaviour that is common to all UI fragments.
AbstractHomeUiFragment	Abstract Class	Abstract class that define basic behaviour that is common to all UI fragments associated to home activity.
RoutePlannerFragment	Class	UI fragment for route planning component.
RouteDisplayFragment	Class	UI fragment for route display component.
RouteAnalysisFragment	Class	UI fragment for route analysis component.
LocationSelectionFragment	Class	UI fragment for location selection component.
NavigationFragment	Class	UI fragment for navigation component.

4.5.4 Presenter Module



A **presenter** handles the **communication between models and views**.

It subscribes to state change in models and update the corresponding views. In reverse, it updates associated models when user events are received.

Name	Type	Description
AbstractPresenter	Abstract Class	Abstract class that provide basic feature and unified interface for subclasses.
RoutePlannerPresenter	Class	Presenter for displaying route planner page.
RouteDisplayPresenter	Class	Presenter for displaying planned route on map in route planning page.
RouteAnalysisPresenter	Class	Presenter for displaying route information in route analysis page.

4.5.5 Decorator Module

A **decorator** adds **presentation logic** to the original data model class. It reduces coupling between components.

Name	Type	Description
Decorator	Abstract Class	Abstract class that provide basic feature and unified interface for subclasses.
NKDirectionsDecorator	Class	Decorator for adding presentation logic to directions objects.

4.5.6 Widget Module

Widgets are **reusable custom view components**. It encourages code reuse and presentation logic separation.

Name	Type	Description
LocationSelector	Class	Custom view component applied in starting point selection and destination selection in route planning page.
NKElevationChart	Class	Custom view component for displaying elevation data as a line chart.
TwoStatedFloatingActionButton	Class	Custom view for floating action button with different display in enabled and disabled state.

4.6 Wear Companion Module

Wear companion module is the logic of the **wear companion application**.

The companion application starts when turn-by-turn navigation is in progress. It **display the navigation advice** and **notify user about important events**.

4.6.1 Core Module

Core module in the wear companion application deals with **lifecycle events**, **communicates with mobile application** and **manages user interface**.

Name	Type	Description
MainActivity	Class	Entry point activity for wear companion application. Deals with **lifecycle events** and **communication with mobile application** during navigation.
NavigationStatePresenter	Class	Presenter for displaying current navigation state.
NavigationStateDecorator	Class	Decorator for adding presentation logic to navigation state objects.
TurnLevel	Class	Enum for representing turn level of a directions object.

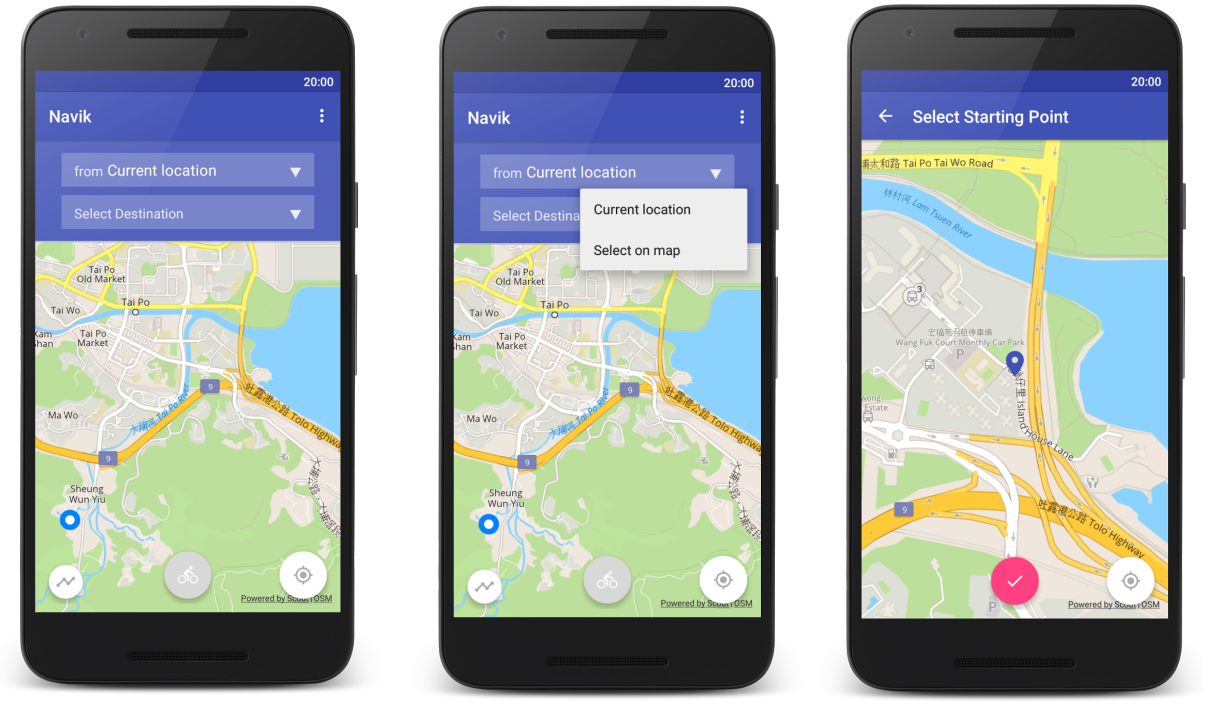
4.6.2 Background Module

Background module is responsible for background communication with mobile application.

Name	Type	Description
WearMessageListenerService	Class	Service that subscribes to messages from mobile application to start the activity.

5 Implementation and User Interface

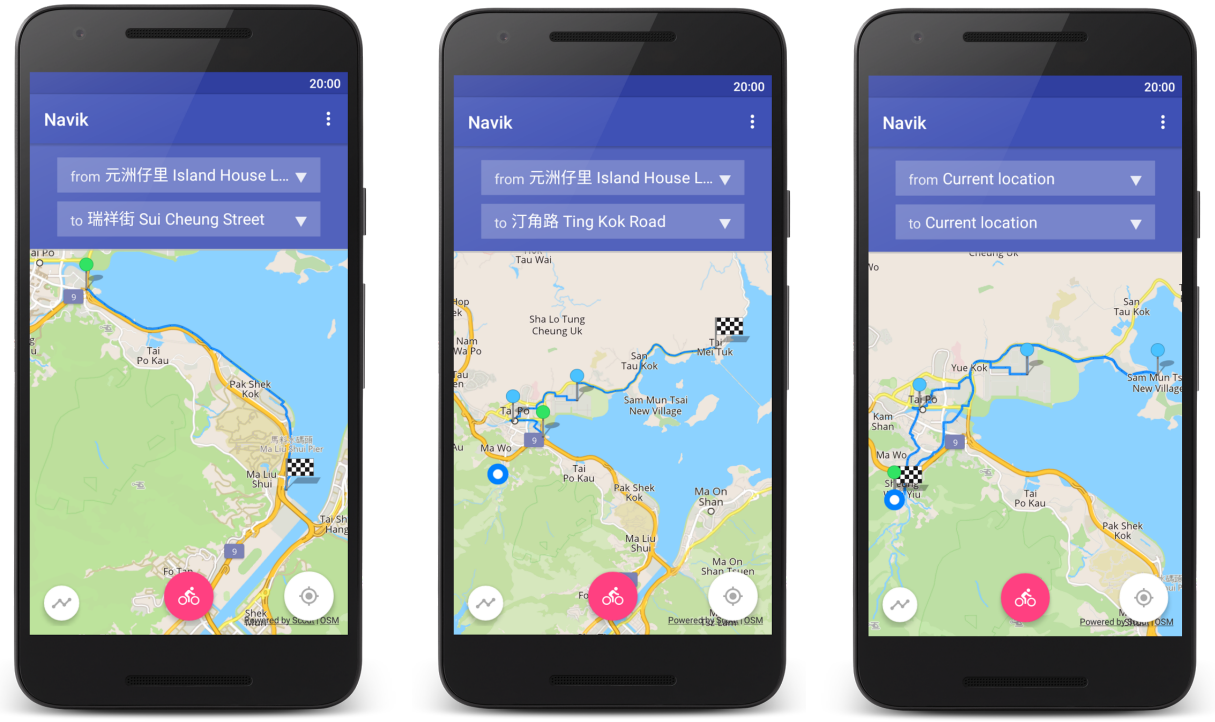
5.1 Core User Interface



The core user interface consists of **one single activity**, **UI fragments** for each page and **UI widgets** for reusable custom views. **UI communications** are mainly achieved by **observer pattern**.

The UI fragments are managed by a **fragment controller**. The fragment controller **controls UI flow** by **swapping UI fragments in and out**, provide **request and result data flow mechanism** and **manage shared UI components**.

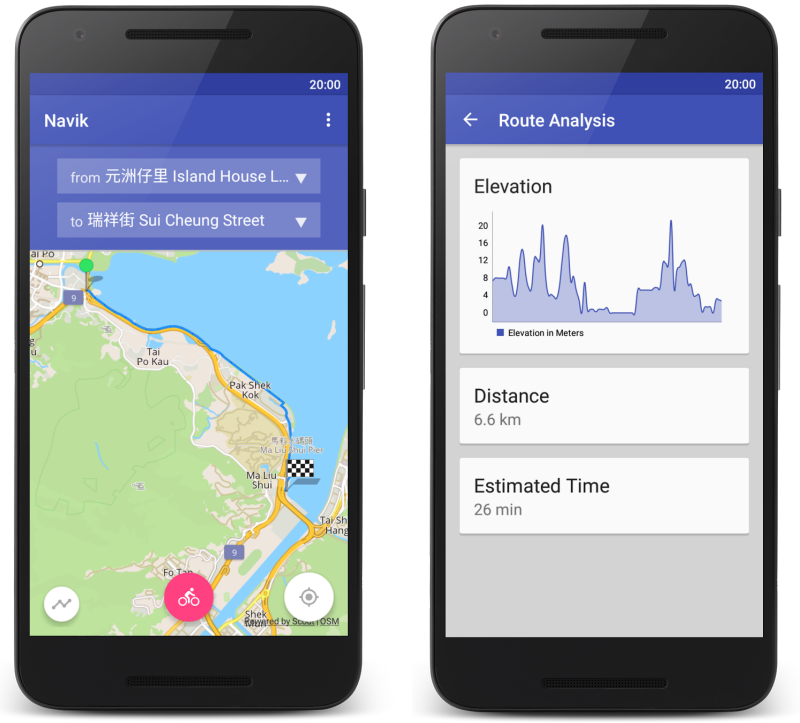
5.2 Route Planning



Users are allowed to do **end-to-end route planning** by selecting a starting point and destination. Planned route would be displayed in the map after route calculation. User are allowed to import **external route** from a **GPX file** as well.

Waypoints could also be added to change the route. **Circular route** allows user to plan a route that goes back to the starting point. It is particularly useful for cycling navigation.

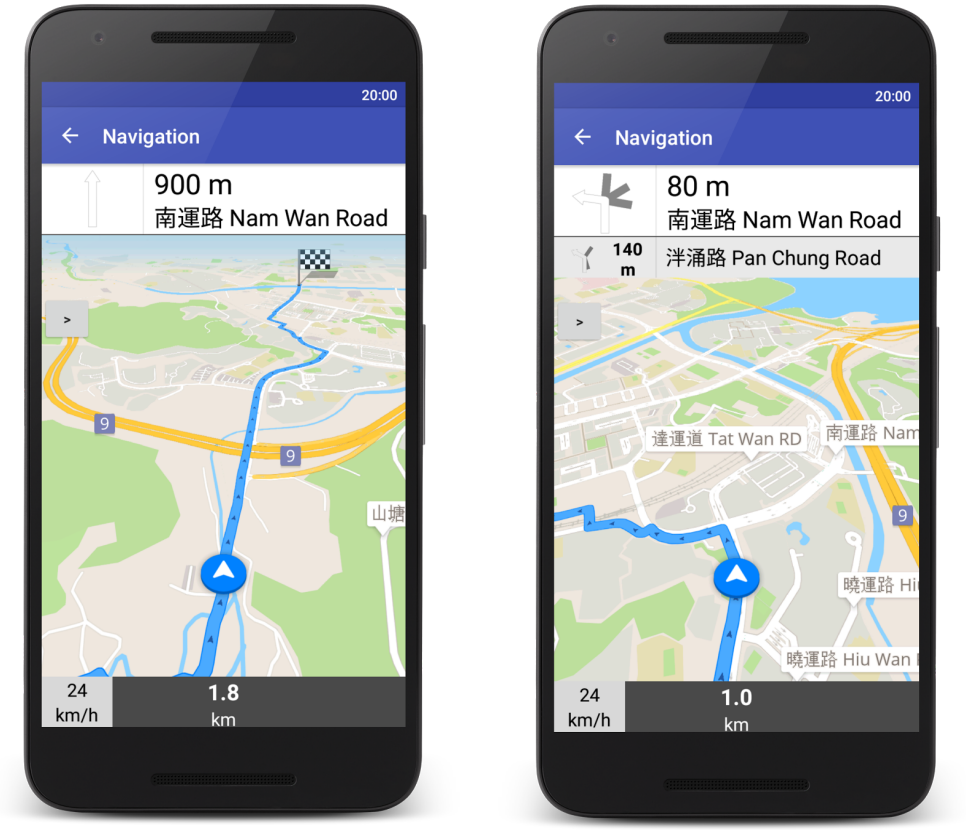
5.3 Route Analysis



Route analysis page provides basic information about the planned route. Thus the user can have knowledge of the route in advance before starting a navigation.

It provides user an **elevation chart**, **distance** and **estimated travel time** of the route.

5.4 Turn-by-turn Navigation



Turn-by-turn navigation allows user to navigate through the planned route. **Real time navigation advices and information** would be shown. **Re-routing** would be done if the user is not traveling on the planned route.

The **next turn** and **distance to next turn** is particularly useful for the navigation. **Current speed** and **distance to destination** are also provided.

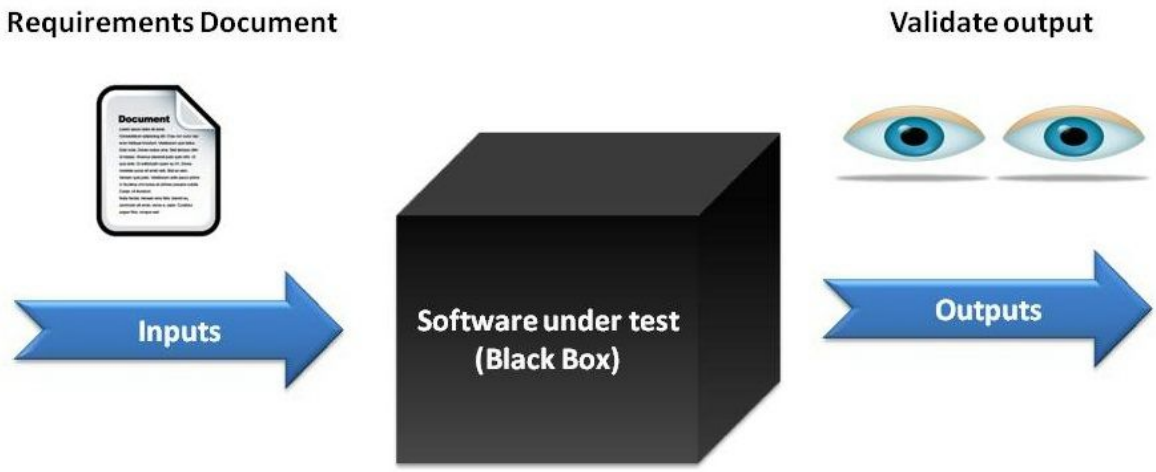
5.5 Wear Companion Application

Wear companion application starts when turn-by-turn navigation is in progress. It **display the navigation advice** and **notify user about important events**.

The **background of the screen changes** as the user **approaches the next turn**. It allows user to know how far away is the next turn easily without reading the text. Furthermore, **critical navigation state changes** would be notified via **vibration notification**. It would help the user determine when to turn without looking at the screen.

6 Testing

6.1 Methodology



Black-box testing techniques are applied. Testing is done without referring to any knowledge to the internal structure or source code. The testing focuses on the **functionalities** of the application.

Test cases are constructed with regards to the **functional requirements** defined as well as the designed **use cases**. **Steps** of the test cases and **expected results** are clearly specified. The **actual results** are recorded after the test case is carried out.

6.2 Testing Environment

Smartphone: Xiaomi Redmi 1S

Smartphone OS: Android 5.1.1 Lollipop (Cyanogenmod 12.1)

Smartwatch: LG G Watch W100

Smartwatch OS: Android 6.0.1 Marshmallow

6.3 Test Cases

6.3.1 Route Planning

6.3.1.1 End to End Route Planning

Steps:

1. Select starting point on the map via the location selector
2. Select destination on the map via the location selector

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.1.2 End to End Route Planning from Current Location

Steps:

1. Select current location as starting point via the location selector
2. Select destination on the map via the location selector

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.1.3 End to End Route Planning to Current Location

Steps:

1. Select starting point on the map via the location selector
2. Select current location as destination via the location selector

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.1.4 Route Plannning with One Waypoint

Steps:

1. Select starting point on the map via the location selector
2. Select destination on the map via the location selector
3. Add a waypoint by long click on the map

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.1.5 Route Planning with Multiple Waypoints

Steps:

1. Select starting point on the map via the location selector
2. Select destination on the map via the location selector
3. Add the first waypoint by long click on the map
4. Add the second waypoint by long click on the map
5. Add the third waypoint by long click on the map

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.1.6 Circular Route Planning with One Waypoint

Steps:

1. Select starting point on the map via the location selector
2. Add a waypoint by long click on the map
3. Select destination as the starting point via the location selector

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.1.7 Circular Route Planning with Multiple Waypoints

Steps:

1. Select starting point on the map via the location selector
2. Add the first waypoint by long click on the map
3. Add the second waypoint by long click on the map
4. Add the third waypoint by long click on the map
5. Select destination as the starting point via the location selector

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.1.8 Import External Route via GPX file

Steps:

1. Select the import external route menu item
2. Choose a valid GPX file to import

Expected Results:

- Route planned and displayed on the map

Actual Results:

- Route planned and displayed on the map

6.3.2 Route Analysis

6.3.2.1 Perform Route Analysis

Steps:

1. Select starting point on the map via the location selector
2. Select destination on the map via the location selector
3. Click the route analysis floating action button to enter route analysis page

Expected Results:

- Route analysis complete and information are displayed on the page

Actual Results:

- Route analysis complete and information are displayed on the page

6.3.3 Turn-by-turn Navigation

6.3.3.1 Perform Navigation Simulation

Steps:

1. Select starting point on the map via the location selector
2. Select destination on the map via the location selector
3. Turn simulation mode on
4. Click the start navigation floating action button to enter navigation page

Expected Results:

- Route navigation simulation starts and complete till the destination is arrived

Actual Results:

- Route navigation simulation starts and complete till the destination is arrived

6.3.3.2 Perform Real Navigation

Steps:

1. Select starting point on the map via the location selector
2. Select destination on the map via the location selector
3. Turn simulation mode off
4. Click the start navigation floating action button to enter navigation page

Expected Results:

- Route navigation starts and complete till the destination is arrived

Actual Results:

- Route navigation starts and complete till the destination is arrived

6.3.4 Wear Companion Application

6.3.4.1 Perform Navigation Simulation with Wear Companion Application

Steps:

1. Connect to the smartwatch with the wear companion application installed
2. Select starting point on the map via the location selector
3. Select destination on the map via the location selector
4. Turn simulation mode on
5. Click the start navigation floating action button to enter navigation page

Expected Results:

- Route navigation simulation starts and complete till the destination is arrived
- Wear companion application starts and displays navigation advices on the screen
- Wear companion application notifies critical advice state changes via vibration

Actual Results:

- Route navigation simulation starts and complete till the destination is arrived
- Wear companion application starts and displays navigation advices on the screen
- Wear companion application notifies critical advice state changes via vibration

6.3.4.2 Perform Real Navigation with Wear Companion Application

Steps:

1. Connect to the smartwatch with the wear companion application installed
2. Select starting point on the map via the location selector
3. Select destination on the map via the location selector
4. Turn simulation mode off
5. Click the start navigation floating action button to enter navigation page

Expected Results:

- Route navigation starts and complete till the destination is arrived
- Wear companion application starts and displays navigation advices on the screen
- Wear companion application notifies critical advice state changes via vibration

Actual Results:

- Route navigation simulation starts and complete till the destination is arrived
- Wear companion application starts and displays navigation advices on the screen
- Wear companion application notifies critical advice state changes via vibration

7 Schedule

7.1 Milestones

Milestone	Incremental Accomplishments
Milestone 1	Smartphone application • Complete **user interface** • Able to **load** and **display** the offline **OpenStreetMap** data
Milestone 2	Smartphone application • Able to **import external routes** via GPX files Android Wear companion application • Complete **user interface**
Milestone 3	Smartphone application • Able to **construct routes** from destinations • Able to perform **route analysis**
Milestone 4	Smartphone application • Complete **turn-by-turn navigation** Android Wear companion application • **Turn-by-turn navigation** display
Milestone 5	Smartphone application • Provide **real-time information** Android Wear companion application • Provide **real-time information**

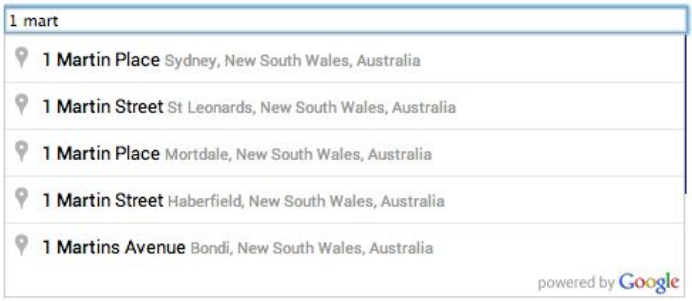
7.2 Timetable

Date	Accomplishments
4 October 2015	Complete **inception phase** • Detailed project plan • Project web page
8 November 2015	Complete **milestone 1**
6 December 2015	Complete **milestone 2**
11-15 January 2016	First **presentation**
17 January 2016	Complete **milestone 3**
24 January 2016	Complete **elaboration phase** • Preliminary implementation
13 March 2016	Complete **milestone 4**
10 April 2016	Complete **milestone 5**
17 April 2016	Complete **construction phase** • Finalized tested implementation • Final report
18-22 April 2016	Final **presentation**
3 May 2016	Project **exhibition**
6 June 2016	Project **competition**

8 Conclusion

8.1 Future Improvements

8.1.1 Search for Locations from Name



User can now select a location for route planning only on the map. It may not be desirable for some use cases. The user may want to **find a location by providing its name**.

A **location searching page** should be added. Hence user can simply enter the location name and select from the search results.

8.1.2 Better Waypoint Management

Waypoint are added by long click on the map in current implementation. It may not be intuitive for some users. Moreover, **rearrangement** of waypoints is not possible.

To enhance user experience and functionality, a **waypoint management page** should be added in the future. Thus allowing user to **add**, **remove** and **rearrange** waypoints in the designated page.

8.1.3 Save and Load Planned Route

For now, users have to construct the route from scratch every time. It is a natural progression to allow users to **save and load planned route**.

User should be allowed to **rename**, **remove**, **load** as well as **categorize** and **export** the planned route.

8.1.4 iPhone and Apple Watch Port

Current implementation only runs on Android devices. iPhone accounts for nearly half of the present market share. It is a reasonable plan to enlarge the user base to iOS.

Additionally, no doubt that Apple watch is more **popular** and **fashionable** than Android smartwatches. An **iPhone and Apple Watch port** would be a brilliant thought.

8.1.5 Publish to Google Play Store

It would be great to publish the application and reach users in the whole world. A **freemium revenue model** would suit the application very well. User would be charged for certain value-added or advanced functionalities.

8.1.6 Online Community

It would be a great idea to build an **online community** for **sharing planned route**. Users are allowed to freely **search**, **upload**, **download** as well as **rate** the planned routes.

8.2 Reflections

Working on this final year project is a **memorable experience**. I would like to **express my gratitude** towards my **supervisor Prof Lau Francis**. He is a nice person as well as offers me a vast flexibility and freedom for the project.

There are certainly both advantages and disfavors for a **one man development team**. It is quite boring and frustrating sometimes working by oneself. However, I can determine the whole design and implementation of the project. It is a also great chance to **practice** my skills and have **advantageous experiments** on various design patterns and libraries.

This application is my **sixth or seventh** Android application as I remember. I can really feel my improvements throughout these years. Software development should be an **enjoyable process**. We **try**, **stumble**, **learn** and **conquer**.

Life should be **a journey of sublimation and self-overcoming**. Embrace yourself.

9 References

1. Garmin Edge 510 - https://buy.garmin.com/en-US/US/business/vitality-members/edge-510/prod112885.html
2. Hammerhead - http://hammerhead.io
3. Incremental model - http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it
4. Dagger - https://google.github.io/dagger
5. Which design patterns are used on Android? - http://stackoverflow.com/a/6770903/2621216
6. Model-View-Presenter Architecture in Android Applications - http://macoscope.com/blog/model-view-presenter-architecture-in-android-applications
7. Strategy Pattern - http://www.oodesign.com/strategy-pattern.html
8. Observer Pattern - http://www.oodesign.com/observer-pattern.html
9. Inversion of Control Containers and the Dependency Injection pattern - http://www.martinfowler.com/articles/injection.html
10. What is Black Box Testing? - http://www.webopedia.com/TERM/B/Black_Box_Testing.html