**Introduction**

With the evolution of smart phones and tablets, maps become ubiquitous in everyday life. Mobile users access the map all the day to do simple queries (e.g., find a nearby restaurant), more advanced queries (e.g., find a shortest path from A to B bypassing toll stations), and even socializing with friends (e.g., check-in to a restaurant). Despite the powerful resources available to new smart phones (e.g., touch screen and many sensors), smart phones only provide a simple interface for viewing and querying the map. However, many users can actually create and edit maps if they are provided with the right tool to use.

In this project, we want to elevate mobile maps applications by providing users with a simple way to edit existing maps. Mobile users usually do not have the time to do complex edit operations such as drawing roads but they can still help by doing simple edits such as tagging buildings or adding points of interests. This project focuses more on the interface part. In other words, the project tries to come up with a mobile interface that is suitable for smart phones and allows users to edit maps. We first give a background of HCI. After that, possible tacit sources and how they are going to be used are provided. Finally, we give detailed results of the experiments we did with the tacit sources and wrap us with results and future work.

**HCI**

Reviewing the document talking about HCI we found they they describe the human computer interaction (HCI) field and how it can be applied in GIS. Basically, HCI is meant to create good a good interface between the human (thinking) and the computer (processing). The interface should be easy to use and expressive. There is a necessary background of the HCI as general that we need to know which includes the basic elements used in this science/art. A computer displays its information on different output devices, so the system designer should be aware of the different output devices and the limitation of each one. Also, the user gives command and data to the device using input devices such as the keyboard or touch screen. Knowing how the user uses each device and which information is suitable for each one makes it better for the system designer to provide efficient interfaces. Another thing is that we should not expect the user to explicitly tell us everything through the input devices. We have to guess what the user really wants to do and provide the correct output with the minimal information from the user. For example, recommendation systems tries to predit what the user liked through his history and the history of other (similar) users. Over the years, the interface has evolved from a text-based interface, to graphical user interfaces that use more advanced input/output devices. Even the graphical user interface evolved drastically from using menus to windows to the latest interfaces on smart phones and other touch devices. All these are about the interfaces as general.

To relate this to our course and our project, we should talk about map interface as the basic element in GIS interfaces. There are specific symbols and techniques to display the information on a map and to take user input from a map. Most display techniques on computer are just a way to mimic what people have been doing on physical maps for centuries. For example, it uses
different symbols, different colors, lines, shapes of different sizes and other techniques to display the information just like it was done before on paper maps. Since the computer is more powerful than a simple paper, there are more advanced techniques for displaying that can be used in computer-based GIS applications. One basic thing is the user of animations to add the time dimension to the map. More advanced techniques display maps in 3d (like Google earth) which takes the user experience to a new level. Taking user input is a little bit tricky with the different input devices available these days. Traditionally, keyboard and mouse gestures were the main ways for the user to input data and commands. New input devices like the 3d mouse, touch screen and kinect sensor open opportunities for new ways to design the interface to make it easier for users to interact with.

Most readings we have reviewed talk about designing the map or GIS interface on a PC or similar machines. Our focus on this project is to build a map interface for the mobile device. There are many papers describing the design of an interface for mobile devices and other touch devices (e.g. tablets). The main differences are the smaller size of the screen and the different input device (touch screen compared to mouse and keyboard). Another important point is the lack of consciousness from users. Mobile users usually use their devices while walking in the street or talking with a friend or even sometimes while driving (despite the fact it is very dangerous, many people do it). So the interface should be extremely simple and descriptive. Most of the work described how to use the interface for querying the map but editing the map on a touch device is not studied well. There are some works about volunteer GIS and we need to build on that work by providing a good and simple interface for users to input data on maps from their mobile devices. We think that this will open opportunities for many applications to collect enormous user data and use it for new applications.

**Tacit sources**

Our main tacit sources will be other applications related to what we are trying to do. We think that the most important application is the editing application provided by open street maps. This is the main application used by volunteers to create the map and edit it as mistakes are found or street maps change. This is a desktop application built in Java and it is quite a good application given the high quality maps created by users for Open Street Maps. We are going to use this application as our model for building a mobile application. First of all, we understand that we cannot beat the desktop application. Our project is not considered as a replacement for the Open Street Maps desktop application. Rather, we consider it as a helper that allows new users to do minor changes to the map according to something the user notices about it. For example, let’s say a user found that his street name is misspelled. It would be very convenient if this error can be fixed rightaway from a mobile device instead of returning home, downloading the application, downloading the corresponding map, editing it and finally uploading the changes. We believe that many users will just ignore the mistake instead of fixing it to avoid this long process.

We are going to start by carefully studying the desktop version. We will list the most important features and functionalities and see which of these we can implement on a mobile device. We
will put in our consideration the limitation of the device we are working on both technical and usability limitations. We will list a few features that we feel can be implemented well on the device and see how the interface should look like for these features. We would like to highlight here that we focus mainly on the interface. We believe there are many other challenges such as handling the massive amount of updates (possibly conflicting) from many users at the server. The focus of this project is to just build the interface.

Here is a list of experiments we are going to do with the desktop application:

- **Installation and configuration**: We need to see how an editing program needs to be installed and configured. This will give us an idea of how a mobile application should do at the beginning.

- **The main interface**: We will examine the home screen of the program and what features are provided in the start screen. We think that the features displayed in the beginning are the most important ones. We will see which of these features are appropriate to have on a mobile device.

- **Downloading the data**: The first step in editing the data is probably to download it first. There must be a navigation or query interface that allows the user to select the part of the world that needs to be edited. We should see how this compares to other navigation/query interfaces on the mobile and if it looks different, we need to study why this difference is important.

- **Examining the data**: Once the data is in the local device, the user is allowed to examine and see the details of this data. Since the user is allowed to edit it, it should have much more details than displayed in a query interface like Google maps. We will study these levels of details and see to which level a mobile user is likely to be interested in. We have also to consider the limited size of the screen at this point as we cannot easily fit everything on a 15" screen to a 4" or 5" screen.

- **A simple edit**: We will perform a very simple edit such as moving a point or editing some labels. We will see how this is done on the desktop device. Which input devices are used? An important thing we will study is how the application controls the accuracy of the user and how it allows for a precise placing of points or labels.

- **Save and submit**: After the user performs the required edit, it needs to be saved and the changes are submitted to the server for revision. We will see how this process is done and how it should look like on the mobile screen.

Other sources we need to check are other mobile maps interfaces such as Google maps and Apple maps. Although these interfaces are only for querying, we can learn from them how a user can interact with a map element on a small screen. There are also some papers about building a map interface on a touch device and we are going to consider all this material before starting this project.
An important application is AutoCAD for mobile devices. This program allows the user to view and perform simple edits to an AutoCAD drawing. We think we can import some ideas from this program by checking how edits are done and what features are available on it.

**Results of tacit information**

We started by going to Open Street Maps website to see how a user can start viewing and editing maps. Although the maps are open for anyone to edit, users must first register a new account for free. This ensures that the system keeps track of who edited the maps. The registration process is fairly simple and requires the user to accept their license agreement. Once a user is registered, she can start editing maps right away using a simple web interface. For more edit options, users can download a desktop application that is more powerful than the web interface. First we review the web interface then move on to the desktop application.

**Web interface**

For simple edit operations, Open Street Maps provides a simple web interface called “Potlatch”. Potlatch runs as a Flash application which somehow limits the platforms that can host it. When the program initially starts, it shows the following home screen.

![Open Street Maps Home Screen](image)

The main part of the application shows a map similar to Google maps. The map contains two main layers. In the background, satellite imagery obtained from Bing is shown to assist users draw the map.
On top of the satellite imagery, the map stored by Open Street Maps is shown as a set of polygons and lines. Each polygon represents a building or an area (e.g., Lake or park). Clicking any of these areas, show handles that can be used to edit this area.

A user can drag any of these handles to move points and adjust the shape. There are also points that represent points of interests (POI) or simple items such as a tree. After the user is finished with editing the map, these changes can be either saved to the web site or discarded (undone).
Open Street Maps also stores information about each building or item to allow their customers to make use of this data in a better way. A user can tag each item (polygon or point) with any number of tags which are stored as extra information about the tagged item. For example, the university recreation center can be tagged as ‘recreation center’ and ‘university’. This indicates that this building is part of the university and it is a recreation center. Open Street Maps has an extensive set of predefined tags categorized to make it easy for users to tag any item.

Potlatch is very easy to use and it does not need any setup or installation process. However, it has a limited functionality compared to what users might want to do with a map. That is why Open Street Maps provide a desktop application that is more powerful in editing.
Desktop editor

For users who want a sophisticated editor, Open Street Maps provide a desktop application that has more editing options compared to the web editor. The desktop version is called “Java Open Street Map” editor, or JOSM for short. The desktop version is a little bit different. The powerful functionality it has comes on the cost of complexity to users. The program itself is straightforward to download and install. When it starts up, it gives the user an option to download the map of an area that will be edited. This can be done in several ways such as pan and zoom in a world map, giving coordinates in latitude and longitude or specifying an address. The following image shows how this screen looks like.

Once the selected area is downloaded, JOSM also gives the user an option to download more layers to assist editing maps.

Add imagery layers?

For the downloaded area, the following additional imagery layers are available:

- USGS Large Scale Imagery
- USGS Scanned Topographic Maps
- OSM - Tiger Edited Map
- MSR Maps Topo

Do you want to add those layers to the Imagery menu?
(If needed, you can remove those entries in the Preferences.)

☐ Do not show again (remembers choice)
After the map is downloaded, it looks like this.

The user can interact similar to the web version “Potlatch”. In other words, a user can edit points of a building or add tags. However, there are more sophisticated tools the user can use to edit the map. The next figure shows how the map looks like when it is zoomed in.

Selecting a building allows the user to edit it by moving point handles.
As shown in figure, there is more information in this edit view. For example, it shows directions on polygon edges. Also the toolbar on the left gives the user a plenty of tools to edit the polygon.

On the right, the user can edit the tags and other information related to the selected item.
The user can edit this information or add new information to the selected building.

**Proposed work**

In this project we propose a mobile map editor that works from the browser. This makes it interoperable from any mobile or tablet device supporting HTML5. We also propose a list of guidelines that is meant to be used as a reference for system designers when building an application for map editing on mobiles. To our knowledge, this is the first project that studies the design and implementation of a map editor on smart phones.

**Mobile map editing guidelines**

In this section, we list our findings for mobile map editing on smart phones. We summarize these guidelines in the following three points.

1. Integrate map viewer with editor.
2. Quick edit and save of changes.
3. Tolerate uncertainty.

**Integrate viewer with editor**
This point means that the viewer and editing interface should be integrated in one application. A typical user that edits a map on a mobile phone, starts as a normal user that requests some service from the mapping application. As users discover errors or missing information in the map, they want to recover these errors right away. It should be possible to fix the error or add the missing information with a few taps on the screen rather than installing a dedicated application and going through a different scenario.

The proposed application solves this problem by providing a unified interface that can be used as a normal mapping application for searching and browsing. With one tap on the edit button, the application lays another layer which provides the edit controls. The user can edit the map very easily and commit the changes, then she can return to the view mode with a single tap on the screen. This motivates users to improve map data right away without forcing them to leave their current session.

**Quick edit and save**
The user must be able to do the changes and save them to the server very quickly. Mobile users usually do not have time to go through a complicated process to make a change. Most of the time, the edit is very simple and can be done quickly. The interface of the editor should be as easy as possible for typical tasks of mobile editors. Typical tasks include modifying tags of a place or adding a point of interest. If the editing application focuses on these small tasks, it can provide greatly optimized user experience.

Our proposed application allows users to edit tags very quickly. Starting from the view mode, the user can change to edit mode, select and object, edit the tags, commit and the change, and return back to the view mode in a dozen of seconds. The user requires exactly five taps to do the mentioned job given that the application starts at the location the user wants to edit.

**Tolerate uncertainty**
Mobile users usually do not pay much attention to the edit done. They might be doing the change while sitting in a café, walking in the street or standing in a line. The application designer should not expect such users to perform very accurate edits. Thus, the application should be able to tolerate these errors by either fixing it or marking it as uncertain.

The proposed application adds a tag to changes done by the mobile editor marking it as ‘uncertain’. This helps users taking it with caution, confirm it, or further fix it.
System design

As depicted in figure, the proposed application follows a layered design where each layer has a well-defined role. The view layer is the initial layer and its role is to allow the user to navigate through the map and, depending on the application, to access some mapping services. The pick layer is the first layer shown when the user moves to edit mode. It shows the borders and handles of objects that the user can select to edit. Once a user selects an object, the object layer shows up to allow the user to perform edits on the selected object. The shown example allows the user to edit tags associated with the selected object. Notice that these layers are stacked on top of each other resulting in a user interface similar to the one shown above.

View layer
This layer is responsible of viewing the contents of the map as a normal user would see it. Typically, this is a web component that shows a user friendly view of the map allowing the user to navigate and browse the map. This usually supports popular touch gestures such as pinch, tap and slide. In the proposed application, the view layer uses an OpenLayer map component with map contents obtained from OpenStreetMap. This can be replaced by any component that allows the user to navigate in the map and provides an API that returns its extents (boundaries) in latitude and longitude. This API is used by the application to obtain map details from OpenStreetMap and draw the pick layer. The view layer can be replaced by other layers such as satellite imagery or Google Maps. Satellite imagery can be used as a reference to help users edit the map while Google Maps can be used basically to contrast the information OSM with Google Maps.

Pick layer
The pick layer shows controls and handles to users allowing them to pick or select an object to edit. This needs to be drawn very accurately on top of the view layer so that handles match with the objects drawn on the map. The main challenge in implementing this layer is the lack of an accurate pointing device on mobile phones and tablets. The typical pointing device for touch screens is the human finger. Compared to screen size, it covers a considerable area of the screen making it hard to guess which object the user intends to select. Keep in mind that the device reports an exact point to the application which is typically the center of where the user taps the
screen. To overcome this problem, the proposed application checks all objects in a circular range around the exact position reported to the application by the device and check all objects that fall totally or in part in this circular range. If only one object found in this range, it is highlighted as selected. If more than object found, a list of all overlapping objects is reported to the user allowing her to choose the object that is really meant to be selected. The radius of this circle is adjusted to match the average size of a user finger. Of course this depends from a device to another depending on screen resolution (dots per inch).

The pick layer is implemented as an HTML5 canvas which supports basic drawing primitives such as line and rectangle. This canvas is used to draw all the handles for objects currently on display. Canvas does not directly support checking the overlap of a circle with object. It only supports checking if a single point lays inside a drawn shape (e.g., polygon). We had to implement our own algorithm to check the intersection of a line segment with a circle and use it as a primitive to detect objects picked by the user.

**Object layer**
This layer displays information about the selected object allowing editing this object. This layer displays on top of other layers. It is designed to occupy only a small part of the screen which allows users to look at the map while editing information. It is also drawn on a semi-transparent background to allow users to see through it.

**Update logistics**
The proposed application is design to run completely on the client side. When committed, all changes are transferred to OpenStreetMap servers through the supported APIs. In this section, we describe a complete scenario of editing the map giving details of what happens in the background. Let us say that Jane is using OpenStreetMaps and she finds a problem that she wants to fix on the fly.

1. Jane opens the application in her mobile browser. On start-up, the application asks for the current location of the device.
2. The browser displays a message to Jane asking if she wants to share her location with this application or not. Jane accepts.
3. The application uses the current browser location to initialize the map viewer.
4. Jane finds a clear mistake in the map. The street shown in the map is misspelled.
5. Jane taps the edit button.
6. The application contacts OSM servers to obtain the structured information of the current view. The extents of the bounding box are obtained from the map viewing component.
7. Once map information is received from OSM, the pick layer is initialized and the data is used to draw objects on the screen.
8. Jane taps the street she wants to modify. The application receives the location she tapped and goes over all objects received from OSM and check if any of them lays
around the location reported to the application. The application finds the street and displays its information in the object layer.

9. Jane taps the text box that contains the street name and fixes the typo.

10. The application detects this change and stores it in a modification list stored in disk in the client side through ‘localStorage’ supported by HTML5.

11. Jane taps the upload button (assuming she has already logged in).

12. The application creates a changeset in the server with Jane’s user id. Then it writes the change stored in the list to this changeset. After that, it closes the changeset as it does not need to write any further updates to it. Finally, it clears the local list of changes as all changes have already been committed.

13. Jane taps the back button. The application hides the pick layer and object layer and returns back to the view layer where Jane can continue her search.

**Conclusion and Future work**

In this project, we study the problem of designing and implementing a map editor on mobile phones. We reviewed current map editors on desktop and came up with standard guidelines to implement map editors on mobile phones. We even go further by implementing a working prototype that meets the proposed guidelines and work on all modern smart phones with an HTML5 compliant browser.

As we think of future extensions to this work, we think of better using mobile sensors to aid editing the map (e.g., GPS and compass). We also study the support of more editing options such as adding new point of interest.