Prototype Design and Development of LizCapApp Mobile Application

Pankaj Singh, Software Engineering MS Student, Arizona State University, Mesa AZ

**Abstract**—These instructions give you guidelines for preparing papers for IEEE Computer Society Transactions. Nowadays, Field Biologists use a mobile application LizCapApp available for both Android and iOS devices to survey data. The app is developed on Capture–Mark–Recapture Data Collection. Though the app serves its initial purpose it has inherent technical limitations. One of them was the static layouts of the screen and input controls are focused on Lizards and similar taxonomies. To increase the user base of the app among the biologist it needs to be generalized which require dynamic input controls. Along with its software support the app needs to be enhanced to accommodate new requirements over the time. As the app program code is developed natively for Android and iOS platforms, which are quite different in their design and architecture, require two different sets of experts to maintain and enhance current apps features. It is financially difficult to hire experts from two different platforms. So we proposed a solution to develop new version of LizCapApp in one of the popular cross platform tools like Xamarin, Titanium and PhoneGap. After careful evaluation of each platform we found Xamarin to be the most suitable for our app requirement. Once the platform was decided, features of the new version of app has been elicited. A prototype of the app and supporting proof of concepts were developed in Xamarin to find solutions targeting implementation level issues of the elicited requirements to pave the way for a new version of LizCapApp in Xamarin.

**Index Terms**—Xamarin, GPS, Capture-mark-recapture (CMR), cross platform, HockeyApp

—————————— ◆ ——————————

# 1 Introduction

Earlier mobile phones were used only to call other mobile or landline devices. With the advent of smart phones, the uses of mobile phones have gone beyond calling. Enhanced software platforms have provided mobile applications to perform different functionalities. Most of the traditional methods/devices of performing activities have been replaced by mobile applications like alarm clocks, maps etc. Since the emergence of the mobile devices, variety of mobile applications have been developed. One such mobile application was LizCapApp, developed for biological field survey. It is very helpful in collecting survey data and synchronizing it with the database for further analysis. As happens with most of the software applications it also requires enhancements and upgradation meeting the current and future requirements. This project targets requirements and technical feasibility for the transition of the current LizCapApp to a newer version to meet its business and technical goals.

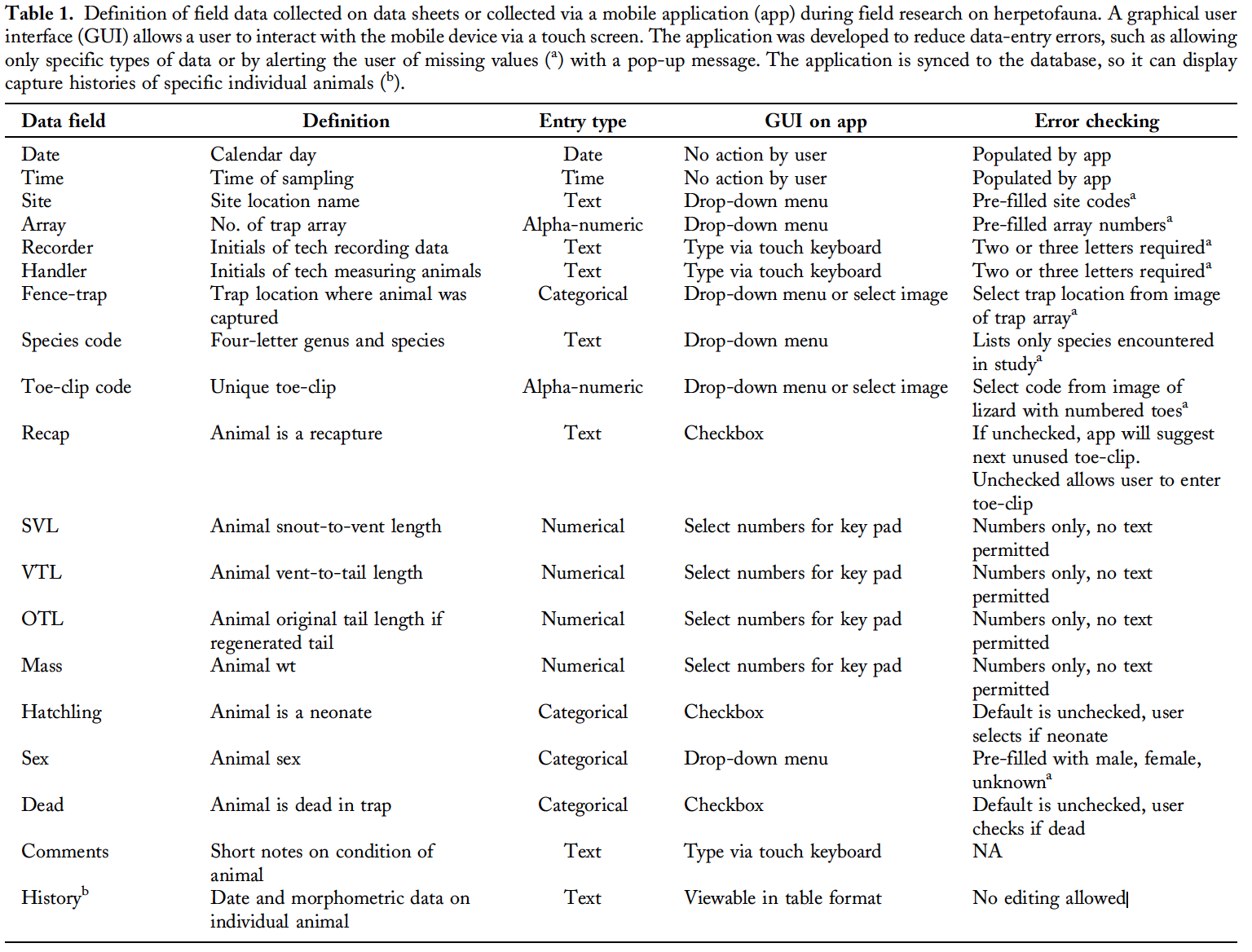
**2 BACKGROUND**

————————————————

1. *Pankaj Singh, Software Engineering MS Student, Arizona State University, Mesa AZ,*  *E-mail: psingh42@asu.edu.*

*.*

Capture–mark–recapture (CMR) methods are often used in the management and conservation of wildlife species to estimate populations and survival [1]. Earlier biologists used paper based datasheets to collect and record wildlife data. Then they come to the lab and feed these values on the computer for storage and analysis. This method has three major modules one at the field for data collection in terms of paper sheets, second in the lab to feed collected data into the lab computers, third to retrieve and analyze this data. This method was easier but error prone majorily at the second stage. So in 2012, Dr. Heather Bateman with the help of Dr. Tim Lindquist, Mr. Richard Whitehouse and Maria M. Gonzalez proposed to use mobile application to collect data in the field and then synchronize it with lab database. They created a mobile application named LizCapApp and experimentally collected data for 2 months [2]. They found that using mobile application reduces errors and was significantly quicker in transferring field data into the lab database. Pre-populated drop down values in the mobile app helped to select correct data. Initial validations of input fields helped to correct data at the field itself rather than correcting values based on memory/assumption in the lab. App has the capability to suggest unique codes to mark every animal, this helped to identify and count new and revisited animals. Mobile app captures detail of the handler and recorder collecting field data. It also recordes the site location where field data has been collected. Mobile app also provides history of a captured animal and can be visited while at field thus enabling to record further analysis of a recaptured animal. Data fields recorded by the app has been summarized in table 1 [2]. Their project was able to prove the advantages of using a mobile application over traditional datasheets for collecting field data in terms of higher data accuracy, less time consumption and better analysis in the field [2]. It is currently used by Dr. Bateman and her students for collecting actual data on Lizards and Arthropods for several research purposes.



**3 LIMITATIONS OF CURRENT APP**

Though the current app was capable of capturing data and

reducing errors in collecting field data it has underlying technical limitations.We have found some of the major limitations which prompted for the need of developing a new mobile application. The requirements of the new app have been elicited and documented in a separate file named LizCapApp\_SRS1.2. Major limitations have been discussed here which are as follows:

**3.1** **Natively developed in iOS and Android**

There are two versions of LizCapApp one natively developed for Android and another for iOS. Both the apps are in production and can be downloaded from iTunes (Apple, Inc.) or Google Play (Google, Inc., Mountain View, CA). Both Android and iOS are the market leaders of the Mobile devices. In today’s market Android and iOS together have a reach to around 99.6% of the smartphone mobile devices [3]. Both the platforms are different from each other in their architecture and design. So while developing an app in either platform, software developers require different skill sets and knowledge to utilize the best from each platform. For example, in Android the app is developed by programming in Java (programming language), while for iOS the developer should know programming in swift/objective C (programming language). This requires hiring different teams for each platform which is very costly and requires a huge amount of time and resources. This increases the maintenance cost of the app over its life cycle. As even a small change or enhancement in app require experts to implement in software code for respective platform. It would have been a cost effective solution to code app in one common programming language targeting both the platforms, thus reducing the size of software developer teams and ultimately cost.

|  |  |
| --- | --- |
|  |  |
| Figure 1. Screenshots of Lizard data from current LizCapApp Android device | |
|  |  |
| Figure 2. Screenshots of Mammal Data and Amphibian Data from current LizCapApp in Android device | |

* 1. **Generalization of the app**

The screens and layout controls of current LizCapApp is highly focused on one Taxonomy of animals i.e. Lizards as shown in Figure 1. Though there are multiple taxonomies available but there input data fields are subset of input fields of Lizards as shown in Figure 2. Now suppose a biologist has to record additionally, color of mammal or change the input field name from mass(g) to weight(kg),( as the input fields are statically implemented within the apps program) he/she has to change the apps’ programming code. As the mobile application has to be used by Biologists, who rarely have understanding of software programming or app development, continuously require support of software developers to modify and enhance apps input fields and their names. To make it generic and usable by large community of Biologist for data collection purpose screens and data fields should be configurable by Biologists without any support of software developers. If the name of the input field has to be changed or an additional input field needs to be added it should be done by the users themselves.

## Locations has to be entered by the recorder manually

In current app Site of the location where the data has been collected has to be selected manually as shown in Figure 3. Though prepopulated values in the dropdown help to get the accurate location value but with current technology of GPS (Global Positioning System) and maps we can eliminate the recording of incorrect manual value. Even the Array field which contains the trap array number at a particular site can be automatically adjusted.

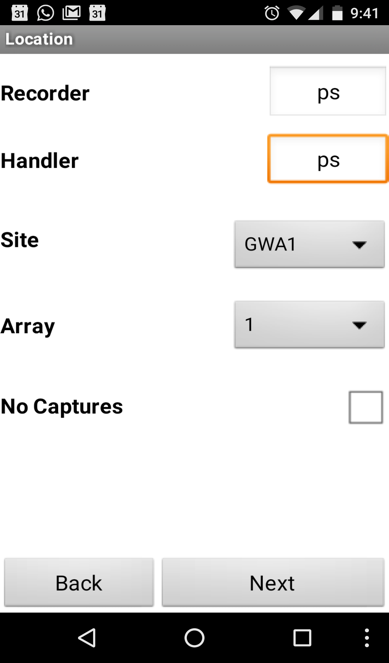


Figure 3. Screenshot of Location selection in current LizCapApp Android

## No bug tracker has been implemented with the app

No software is absolutely bug free, there are always more or less bugs. If the app does not perform naturally, it crashes or gets hanged while recording any data value it needs to be reported by the user to software maintenance team. Most of the time user forget to report or if one reports this unusual behavior, is unable to describe the bug they faced while using the app, even if one describes, the description is not technical in nature. So the technicians are unable to understand the cause of the actual bug or unable to reproduce the bug in the app. There should be an automatic bug reporting tool implemented within the app which can store the state of the app at the occurrence of the bug, thus helping the maintenance team to reproduce and fix it quicker.

## UnSynced History Data cannot be edited

Once the field data has been collected it is stored in the local database of the mobile app and can be seen in the “UnSynced History” module of the app. If there are any errors or inconsistencies in this data which needs to be edited, there are currently no provision to do in the app. The only way is to update it to the Lab database and then correct it there. This correction causes extra effort and time to be spent in finding the inconsistent collected data and do correction as the Lab Data is much larger than recent collected UnSynced data on the app. The effort becomes many fold when multiple mobile devices have such issues.

**4 PROPOSED SOLUTION**

After due deliberation and discussions, we proposed feasible technical solutions for each of the problems/limitations stated above. Even while proposing a

newer version of the app we documented its requirement specification in a separate document named Software Requirement Specification. Solution for each of the limitations are discussed below:

* 1. **Natively developed in iOS and Android**

This was one of the major limitation to be taken care while

developing newer version. We have to find a common language/tool to develop mobile which can run in both iOS and Android. Many companies have launched their platform/tools to develop mobile apps, which provide special feature to develop once and run anywhere capabilities. These tools/ framework/platforms uses a single technology to develop application/s for multiple environments and are generally referred as “Cross Platforms”. We have explored three most popular crossplatforms available in the market i.e. PhoneGap, Titanium and Xamarin on the technical and business needs of the newer version of LizCapApp. We tried to develop sample applications containing most essential user interface components like Input Text fields, Spinners, Buttons, MySQL (SQLite) File upload/download, Images, Connection to Wi-Fi for each platform and uploaded sample applications [Github repository](https://github.com/PankajSingh-ASU/HybridPlatformDemos). Each platform has been evaluated on the following features with their decreasing priority:

*Pricing*: We evaluated free version and trial version of the platform only.

*Native Mobile Features:* Does the platform provide device hardware support? Can they make use of device hardware and optimize applications using native Android or iOS features? After developing an application how close do they resemble to the same app developed in native platforms. Platform has been evaluated for their ability to use native features of the mobile (Android & iOS) devices.

|  |  |
| --- | --- |
| Figure A Initial screen of the app | Figure B Screen to capture Handler and Recorder and site location |
| Figure C If there are no animals have been captured for analysis record the status of the trap. | Figure D Taxanomy of the animals to capture their details |
| Figure 4. Screenshot of LizCapApp on iOS device developed in Xamarin (Replicated screens from original app written natively) | |

*Learning Curve:* Availability of the tutorials, documentation of APIs and prerequisite learnings.

*Community Support:* Do platforms provide developer forum to discuss bugs and errors if yes, how efficient those forums are.

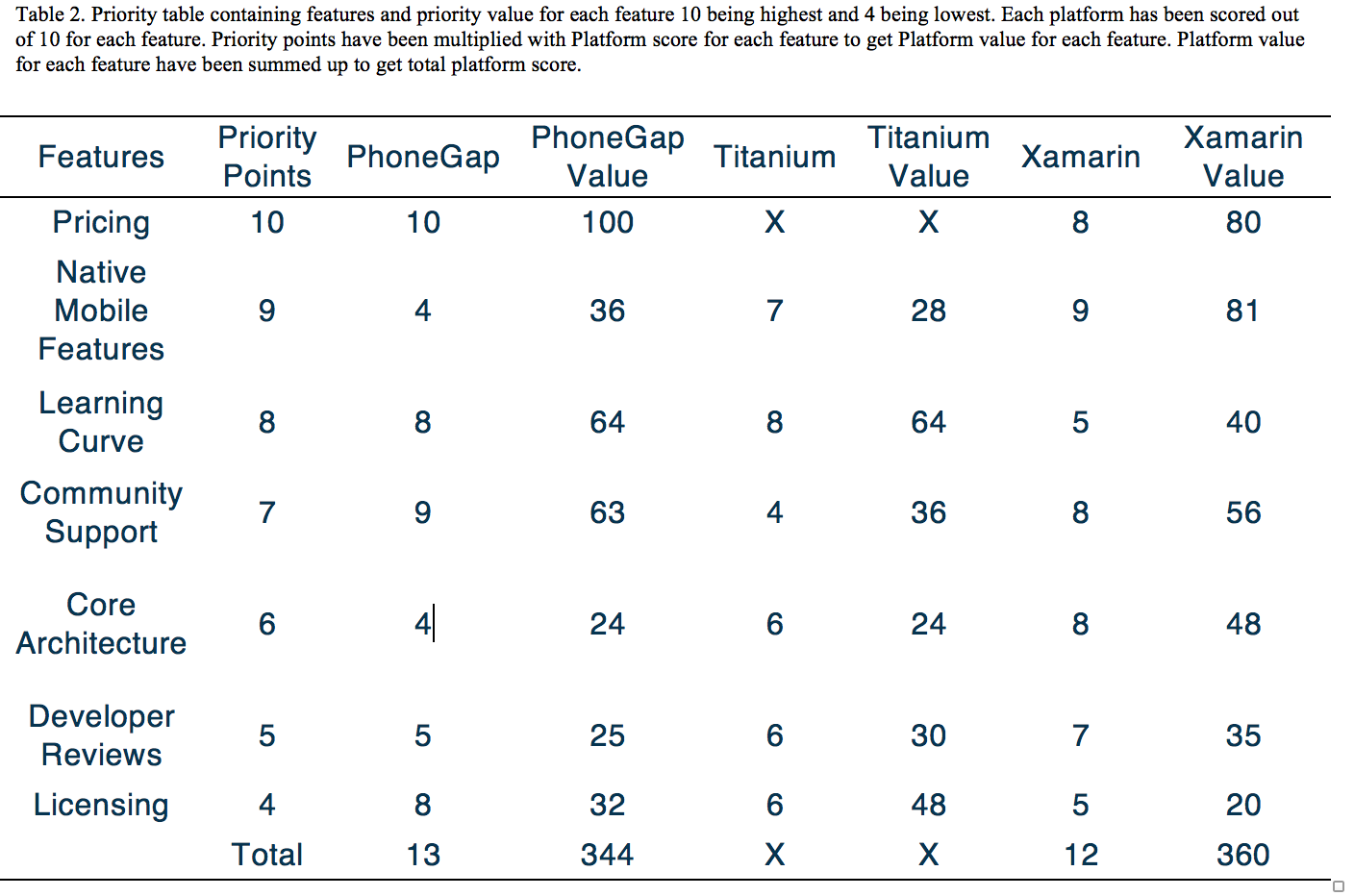
*Core Architecture*: Flexibility of the directory structure of app in that platform.

*Developer Community Reviews*: How many and how strongly developers recommend this platform [5], [6].

*Licensing*: Is the platform available to student/educational institution for free to publish their applications to the marketplaces like iTunes for iOS and PlayStore of Android.

We have used priority table to evaluate each platform on required features and summarized in Table 2.

Each feature is given a priority value for example pricing with max point of 10 and licensing with least of 4. Each platform was given a score out of 10 for each feature. Scored points were multiplied with the priority value and summed them up the to get the final score. Detailed report can be seen in the Appendix A. We found that Xamarin is the most suitable platform for our needs with the maximum score of 360. While the Titanium platform is not free for development it has not been considered. Once Xamarin has been selected, we have developed a prototype application in Xamarin, named LizApp, replicating few screens of current LizCapApp as shown in the screenshots in Figure 4. This application has been uploaded in github along with other codes.



* 1. **Generalization of the app**

Since we are not going to modify lab database or web framework accessing lab data base, we have to find a way to generalize the mobile app with mobile app code itself. In current architecture of the app the lab database is synchronized with the mobile app which means in mobile app we get a copy of lab database in the form of SQLite database (local database of mobile app). After discussion with biologists and our online research we categorised that most of the input data collected belongs to the following data types:

1. Integer or Number
2. Float or Decimal or real numbers
3. String or Text
4. Date
5. Boolean or true/false
6. List (dropdown list)

The data type of columns in a table in SQLite database correspond to these data types. When programming cross platform mobile app with Xamarin we have the privilege to write common logic code or database access code as a common code and design or screen layout code for Android and iOS separate code. So we proposed using SQLite database we should dynamically generate input controls and screen layouts. With Xamarin and in Android mostly SQLite is used to store and retrieve local data values. In most of the cases database schema is already known to the developer which he/she uses to design Object relational mapping to access SQLite Database. In Xamarin you have to include external package to get the capability to access SQLite. There are multiple SQLite packages are available in Xamarin market place add SQLite.Net PCL v(3.1.1) for individual platforms and SQLite.Net.Async-PCL for common code [8]. There are multiple blog posts add SQLite which can be followed but in all the cases they create a new local database during installation. Here we have to synchronize local database with apps SQLite so we have restriction to use this same local SQLite. The best way to perform is after synchronization put the local database in assets folder in Android app. Create a stream to read this file from assests folder a nd create a SQLite db on the Android app device path [9].

Once it SQLite is setup we can use it for our data access and dynamic UI (User Interface) generation. Using the SQLite query [7] we get the column data types of a given table along with cid (unique for every column) in a table. The query gives cid, column name, column data type, not null, default value and primary key for every column of the table queried shown in Figure 5. We created a class with above attributes to access column details through SQLite. Next we can simply return the C# object containing column data type details to the presentation layer of respective environment (iOS or Android).

*<pragma table\_info(‘tableName’) >*

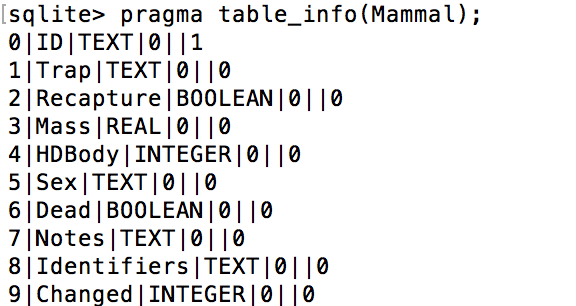
**

Figure 5 pragma query run on SQLite console with its result

|  |  |
| --- | --- |
| Figure A. Number keypad to input textfield value integer values | Figure B. Check boxes to enter boolean value |
| Figure 6. Screenshots of int and boolean input controls generated dynamically on the SQLite column data type for LizCapApp written in Xamarin | |

We have developed a sample application named SQliteTest with Android screens in Xamarin to test feasibility of our idea. Next we matched column data types with the corresponding UI controls as represented in the Table 3.

|  |  |  |
| --- | --- | --- |
| Table 3 Data type matching with UI control | | |
| Data Type | UI control | Keyboard type |
| int , integer | EditText  Box | Android.Text.InputTypes.  ClassNumber |
| Decimal, real | EditText  Box | Android.Text.InputTypes.  NumberFlagDecimal |
| Text, string | EditText  Box | Android.Text.InputTypes.  ClassText |
| Boolean | Check box | Not required |
| Date, datetime | DatePickerDialog | Not required |

While saving the input values collected from the user on the mobile screen takes the reverse route. Depending on the data type user is presented a keypad/UI control to enter values (class DynamicUICreator) shown in Figure 6 and Figure 7. Once he clicks on save the control reads the values from UI controls. Depending on the UI control type like for numbers/int value would be read by using text property of EditText while for Boolean values selected property of checkbox has to be used (getInputValues() method of class InputStore ).

A dictionary is created with the column cid as key and UI input as value. This dictionary would be passed to the Data Access layer of the app then an insert query is created with column names and column values are set from dictionary values (method saveTableValues() of class DatabaseAccess ). Creating a drop down list for selecting a prepopulated list require more deliberation as there is no data type called list for column types in SQLite. There were two logical solutions proposed:

1) to read the distinct values of the column if there were limited count say 5, create a dropdown list as UI control, corresponding to this column. But this solution has many limitations like if this is a new table where less than 5 rows are entered all the columns would be considered a drop down list. Or if the drop down list values contain more than 5 values like days in a week.

2) Create a table containing the possible values of the dropdown list and reference this list column as a foreign key in the original table. For example, while entering Amphibian data user needs to select fence trap, this fence trap column of Amphibian table needs to be a foreign key referencing a column trap in FenceTraps table. There was no easy way in SQLite to find out which column is a foreign key in a table. Moreover, finding the referenced column and referenced table by the foreign key increases the complexity of the solution.

We still have to figure out and finalize the best way to create dropdown lists for an input field without modifying database.

## Automatic Locations Configuration

With the help of GPS technology present in mobile devices we can find the current location of the user. It will return latitude and longitude of the users’ location. Since all the sites are stored with their latitude and longitude value, we can calculate the distance between the site and the users’ location. Least distant site from the user will be the site selection. In this way we will be able to automatically record site of the data collection. Determination of location of a mobile device depends on GPS, Cell-ID, and Wi-Fi but they have their tradeoffs of accuracy and power consumption [4]. Google suggests using ACCESS\_COARSE\_LOCATION or ACCESS\_FINE\_LOCATION permissions. We will recommend using ACCESS\_COARSE\_LOCATION which though did not provide less accurate location but it uses less battery consumption and being the sites far away from each other nearest site can be accurately calculated.

## Bug tracker

There are a number of Mobile App crashing tools available in the market. Some of the most popular ones are Crashlytics, Instabug, HockeyApp, Parse etc. Crash reporting tool contains two major components: one reporting library and second a server-side collector. Reporting library is deployed with the app in the mobile device whose work is to prepare details about the crash. While the server-side collector collects details of the crash from all the reporting library and present in a meaningful way [9]. Xamarin provides Xamarin Insights to track and report app crashing data. HockeyApp is another tool provided and owned by Microsoft Inc. which can be signed up with students account or work/school account. They provide a free plan offering unlimited distribution, crash reporting, and feedback for 2 apps [5]. This will be sufficient for newer version of LizCapApp.

## UnSynced History Data cannot be edited

Currently UnSynced history can be seen in a linear tabular format in a screen in LizCapApp. On click of any cell value in the table a screen navigation should be created to navigate to the data collection screen with prepopulating input field values by reading values already stored in SQLite database. Major advantage of this will be to correct the data when the animal is “in the hand” instead of correcting the data back at the lab.

|  |  |
| --- | --- |
| Figure A. AlphaNumeric keypad to input textfield values | FigureB. DatePicker dialog to select date for captured date |
| Figure 6. Screenshots of text and input controls generated dynamically on the SQLite column data type for LizCapApp written in Xamarin | |

**5 FUTURE WORK**

This project provides the technical feasibility and implementation level details for the newer version of LizCapApp. Development of a fully-fledged new LizCapApp in Xamarin platform, accommodating the features elicited in Software Requirement Specification1.2, will be undertaken next. Dynamic UI control generation from the SQLite database will help in generalizing the LizCapApp for wide variety of field surveys. It will expand the user base including biologists surveying different taxonomies. Even it will help to generate future mobile applications where the static layouts are not preferred and controls needs to be customized by the end users and not the developers. Best way to implement drop down lists needs to be figured out. There will be many other technical challenges to be addressed and solutions have to be figured out during the actual development of LizCapApp in Xamarin. Evaluation of cross platforms can be used to evaluate suitability of the best platforms for future development of mobile applications.

**6 CONCLUSION**

The major goal of the project was to develop a prototype mobile application for field surveys whose development and maintenance should be mobile platform independent. We have developed several prototypes for the evaluation of the platforms. We have identified and evaluated Xamarin to be the most suitable platform for the development of a new field survey mobile application. We developed a prototype to replicate screens of current app in Xamarin to showcase its usability. Another prototype for dynamic generation of UI controls have been developed to find out the implementation details required. Most of the known requirements of the new app have been analyzed and their solutions have been proposed. This project would smoothen and accelerate the programming of the new mobile application for field survey.

**Acknowledgment**

I sincerely thank Mr. Richard Whitehouse for mentoring and guiding throughout the project with his expertise and knowledge. We also thank Dr. Heather Bateman for providing perspectives of Biologists/end users and providing her valuable inputs in all discussions and meetings. We thank Dr. Lindquist for providing technical guidance for developing prototypes.

**References**

1. Sears, A., and Y. Zha. 2003. Data entry for mobile devices using soft keyboards: understanding the effects of keyboard size and user tasks. International Journal of Human–Computer Interaction 16:163–184.
2. Heather L. Bateman, Timothy E. Lindquist, Richard Whitehouse and Maria M. Gonzalez 2012. Mobile Application for Wildlife Capture–Mark–Recapture Data Collection and Query. Wildlife Society Bulletin; DOI: 10.1002/wsb.322
3. Egham, U.K., February 15, 2017, press release.

<http://www.gartner.com/newsroom/id/3609817>

1. Location Strategies

<https://developer.android.com/guide/topics/location/strategies.html>

# [5] PhoneGap vs Titanium vs Xamarin. <http://stackoverflow.com/questions/17249500/xamarin-2-0-vs-appcelerator-titanium-vs-phonegap>

# [6] PhoneGap or Titanium or Xamarin. <http://www.cygnet-infotech.com/blog/phonegap-or-titanium-or-xamarin-which-cross-platform-should-you-choose>

[7] Pragma statements

<http://www.sqlite.org/pragma.html>

[8] Xamarin – Using SQLite.NET Async with PCL

<https://mcnextpost.com/2016/05/30/xamarin-using-sqlite-net-async-in-pcl/>

[9] How can I Access an already existing SQLite database in mono for android

https://forums.xamarin.com/discussion/4238/how-can-i-access-an-already-existing-sqlite-database-in-mono-for-android

[10] [Overview of iOS Crash Reporting Tools:](https://www.raywenderlich.com/33669/overview-of-ios-crash-reporting-tools-part-1)

https://www.raywenderlich.com/33669/overview-of-ios-crash-reporting-tools-part-1

**Supporting Information**

All the sample applications created/analyzed to evaluate cross platforms documents have been uploaded in github repository at [Github repository](https://github.com/PankajSingh-ASU/HybridPlatformDemos)<https://github.com/PankajSingh-ASU/HybridPlatformDemos>

All the supporting documents have been uploaded to Github Repository <https://github.com/PankajSingh-ASU/SER-593-AppliedProject>

Software Requirement Specification document LizCapApp\_SRS1.2 <https://github.com/PankajSingh-ASU/SER-593-AppliedProject/blob/master/Documents/LizCapApp\_SRS1.2.docx>

LizApp