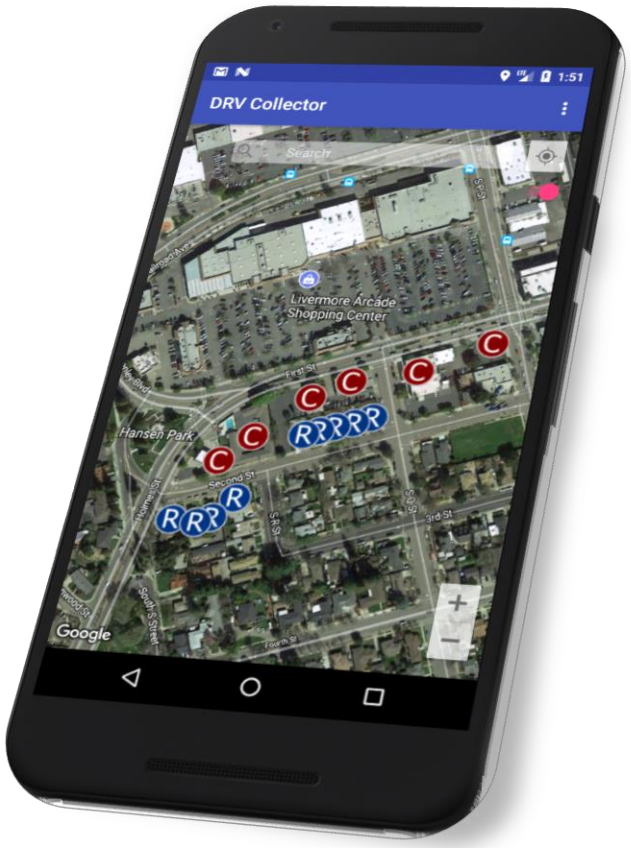
Depreciated Replacement Value Collector Application



James Carney

GEOG 576-070

August 20, 2017

Abstract

In pursuit of dwindling state and Federal infrastructure program funding, small suburban and rural communities must self-perform detailed natural hazard risk assessments to quantify their risk of impact from floods, earthquakes, hurricanes, and other hazards. Risk of damage to structures and their contents within the community typically represents the largest share of risk, in terms of dollar value.

To be eligible for construction or mitigation funding from Federal agencies such as the Corps of Engineers or the Federal Emergency Management Agency, these damages must be estimated in terms of Depreciated Replacement Value (DRV) of each structure using primary data, a level of detail not often available from published market value reports or the local tax assessor's office.

The purpose of this project was to prototype an open-source field data collection mobile application for the gathering of structure characteristics necessary for estimation of DRV on a building-by-building basis. Key tools included a PostgreSQL spatial database, an Apache Tomcat Java Servlet, and the Android operating system.

The prototype shows that such an application has strong potential to provide a low-cost alternative to existing commercial GIS suites, a savings which may be especially attractive to smaller communities without existing GIS departments or data for their community.

Author

James Carney has worked in the water resources engineering and environmental consulting industry for about ten years. He specializes in risk assessment for natural hazards, and benefit-cost analyses in accordance with requirements for participation in state and Federal mitigation programs.

Contents

Tables

Figures

Research Question

Can a purpose-built field data collection mobile application provide more effective and efficient capabilities in support of quantitative economic risk analyses than available tools for small suburban and rural entities?

1. Introduction

1.1. Background

Communities across the U.S. are subject to risk from natural hazards, including all manners of flooding, earthquakes, hurricanes, and more. The existing and anticipated effects of climate change, which in general are expected to exacerbate the frequency and severity of natural hazard events over time (1), have introduced a new layer of uncertainty in the establishment of acceptable risk levels for community decision makers. Coupled with aging infrastructure and inadequate Federal budgets (2), communities must pursue grant and mitigation funds in an increasingly competitive arena.

Quantitative economic analysis procedures like those required by Federal agencies like the U.S. Army Corps of Engineers and the Federal Emergency Management Agency are representative of most Federal programs. Because these agencies must prioritize available funding among projects nationwide, a detailed and explicit tabulation of risk is required to compare potential projects to each other in a consistent and equitable manner.

For many hazard types and communities, risk to private and public buildings, including direct structural damage and loss of the building's contents, accounts for the largest share of potential damages resulting from a hazard event. Development of detailed risk estimates for structures and contents requires primary data. For most studies, properties must be valued in terms Depreciated Replacement Value (DRV), equivalent to a building's replacement cost less any deprecation (3). This is different from assessed value and market value, which are adjusted for tax exemptions or reflective of other economic influences, respectively. To develop estimates of DRV, specific structure characteristics must be collected, such as property type, square footage, construction type, and overall condition, to generate a value per square foot for each structure.

1.2. Problem

To pursue mitigation or other Federal funding for natural hazard risk reduction, communities must often fund initial studies themselves. In suburban and rural areas, this is a large expense, either in terms of additional internal staff or the hiring of a contractor. For communities with limited or no existing GIS information available, substantial time and money may be expended compiling and parsing datasets from multiple departments (that may not ultimately contain useful information), identifying data gaps, and then performing supplemental field investigations. For these communities and studies where cost and schedule are critical to pursuit of external funding sources for risk mitigation, there may be an opportunity for lightweight and specific mobile applications to provide more efficient data collection assistance than standard commercially available tools.

The remainder of this report documents the design, development, and testing of a prototype mobile application for collecting the necessary structure information to rapidly develop a detailed and defensible estimate of the DRV of properties at risk within a community.

2. Research and Design

The prototype application developed in this project was named Depreciated Replacement Value Collector (DRVC). This section describes the design and implementation of DRVC.

2.1. Existing Workflows and Applications

2.1.1. Google Earth/Maps

The current standard workflow for collecting structure data necessary to estimate DRV relies heavily upon Google Earth and Google Maps Street View imagery. This approach works well in areas where this imagery is high resolution. However, the workflow breaks down for areas where Street View imagery is old or unavailable. In these cases, supplemental field data collection is still required.

2.1.2. ESRI Collector for ArcGIS

ESRI's Collector for ArcGIS is the mainstream commercial product which could provide much the same functionality as is prototyped in this project. Collector is highly functional and includes support for working offline, supports multiple device platforms, and is compatible with external GPS positioning hardware. For planning-level studies performed by companies in the environmental consulting industry, where positional accuracy of a mobile phone's GPS radio is more than sufficient, use of Collector is common. It has largely replaced use of dedicated GPS loggers such as those available from Trimble. This is similarly true for large communities with their own GIS departments. Once already bought into the ESRI ecosystem, use of Collector is likely the right choice. However, many smaller communities do not have the staff or time to build internal capability, and a project-specific solution may be preferable.

2.2. Design Goals

DRVC is intended to assist with collection of field data collection. The application will use a mobile device's GPS to locate the user on a map and allow creation of structure points. Fine adjustment to structure location should be possible. Users should be able to categorize and describe each building by selecting from dropdown lists (to ensure data consistency). Users should also be able to see the data previously collected for other buildings, to help reduce variation in selections for similar buildings. To support data exploration, query functions will be included based upon the property characteristic fields. The mobile device's camera should be accessible and photos should be able to be linked to structures in the database.

2.3. Development Tools

The DRVC application was built using the following tools:

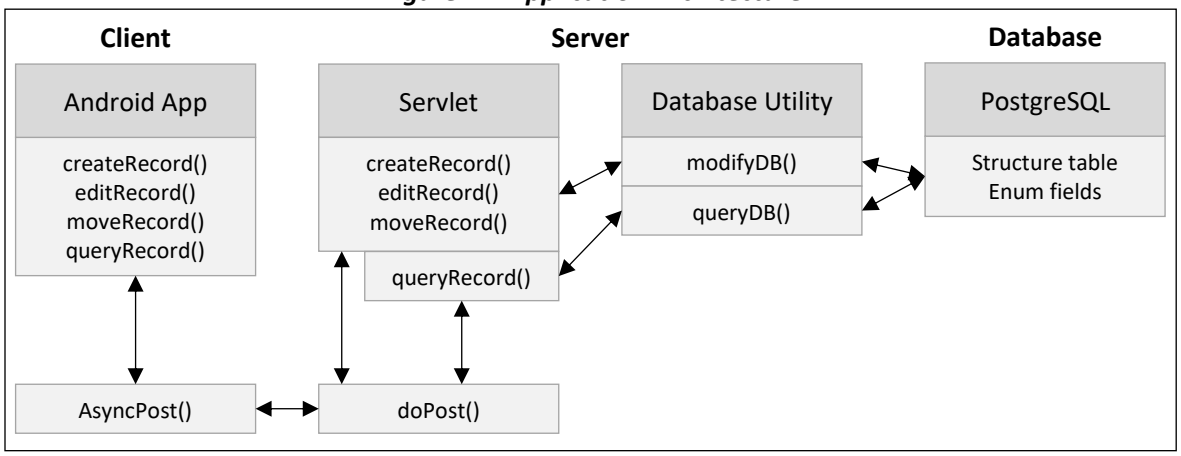
- Database Management System: PostgreSQL instance with PostGIS extension; the opensource Postgres by BigSQL distribution was used for deployment in the Windows environment (4)
- Web Server: Apache Tomcat 7 web server running a Java Servlet (5)
- Client: Android OS mobile application, including use of the Google Maps API
- Galaxy Note 4 for live testing of the application

The IDE chosen for development was JetBrains IntelliJ (6), which provides support for Apache Tomcat as well as all the features of Android Studio. To make the database temporarily accessible over the internet during testing, a free No-IP account was used to redirect requests to the servlet port (7).

3. Implementation

The application's architecture was designed to include a client layer, a server layer, and a database layer. In the prototype application, the database layer and the application server layer are run from the same machine, though this is not a requirement. Figure 1 summarizes the request and response architecture for the DRVC application. The database layer will store field data and accept create, update, and read requests from the server layer. The server layer will consist of a database utility which will connect to and communicate with the database using SQL commands, and will include a Java servlet for converting Java commands to SQL commands, and converted query results to JSON Arrays to be passed to the client. The client layer will consist of an Android application. Subsequent sections provide detailed descriptions of each layer, including the methods referenced in Figure 1.

Figure 1 – Application Architecture



3.1. PostgreSQL Database

The DRVC prototype application uses a spatial database called *drvcollector*. The PostGIS extension provides spatial functionality. DRVC uses very simple database structure, including a single main table named *structure*, an autoincrementing ID field, a geometry field, 13 enumerated string fields, and a field for a custom note. The enumerated fields serve as a data dictionary for users collecting field information. Each field's purpose is described in Table 1.

Table 1 – Field Descriptions for Table "Structure"

Field	Description
id	Auto-incrementing primary key field. Used as the name of the structure in the client application as well.
structurecategory	Structure's general use type as one of Residential, Commercial, Industrial, Public, or Agricultural.
structureoccupancy	Structure's specific use type, such as "Single-family Residential, Detached" or "Restaurant, Fast Food." A general list of about 80 occupancies are included in DRVC. Facilitates later estimation of a structure's content value. Content Value refers to the value of all non-structural items in a building – tenant improvements, furnishing, inventory, etc.

Field	Description
vacant	A simple Yes or No field to denote whether a structure is vacant. For study areas where hazards have occurred recently, this flag ensure that damage modeling does not overestimate losses by assuming all structures have contents.
firstfloorarea	For selection of an estimated range of square footage, such as "1,000 to 1,500." Facilitates validation of specific square footage information retrieved from other sources, such as parcel attribute or tax assessor information.
constructiontype	Categorizes the type of materials used to build the strucure, which helps define a value per square foot. Includes categories such as Masonry, Wood Frame, Steel Beam, etc.
constructionquality	Characterizes the quality of the structure in terms of the choice of materials, features, and finishes that would affect its value per square foot. For example, a wood frame house can range from Cheap to Excellent, where Cheap is four walls and roof, and Excellent is a custom designed luxury home.
structurecondition	An estimate of a structure's current condition relative to when it was new. Facilitates an adjustment to value based on depreciation.
stories	Notes the number of stories
basement	Notes the basement type if present
garage	Notes the garage type if present
staircount	Field technician counts the number of stairs between a structure's ground level and the first finished floor. Facilitates specification of foundation height in flood modeling software, which provides a higher resolution result than only using ground elevation, which is typically retrieved from the best available bare earth DEM.
riserheight	Default value of 7 inches for stair/step risers. Field technician may adjust this value for non-standard stair riser heights.
fieldnote	Custom text entry field
geom	Allows storage of the geographic location of each structure point

3.2. Apache Tomcat Servlet

To provide for communication between the mobile client and the database, a Java servlet was created and run on an instance of Apache Tomcat 7.0.78. In addition to the Tomcat libraries, the following external libraries were imported into IntelliJ to support communication with the *drvcollector* database: java-json.jar, postgis-jdbc-2.1.3.jar, and postgresql-42.1.1.jar. The servlet was created in its own project directory, and includes two classes to manage the connection and requests between the client DRVC application and the database. The first class is called *DBUtility*. It includes three methods:

- **connectDB**: create connection to the database:
- **modifyDB**: submitting of SQL statements to the database without returning a result
- **queryDB**: submitting queries to return a result

The second class is *HttpServlet.* For this project requests from the client are handled in the *doPost* method, which checks the incoming request for a coded value and determines which of the other four methods to call. In each of the HttpServlet methods, data in the request is converted to match the field names in the *drvcollector* database and format a SQL statement. The four methods include:

- **createRecord**: Converts input data into a SQL INSERT command, then passes this string to the *modifyDB* method of *DBUtility*. Used for creating new structures in the database.

- **editRecord**: Converts input data into a SQL UPDATE command, then passes this string to the *modifyDB*. This method requires a structure id be specified to identify which record to update.
- **moveRecord**: Takes as input only a structure id and latitude and longitude, creates and passes a SQL UDPATE command to the *modifyDB* method of *DBUtility* to modify a structure's location.
- **queryRecord**: Converts input data into a SQL SELECT command, then passes this string to the *queryDB* method of *DBUtility*, and then converts the result received from the database into a JSON array to be sent back to the client.

3.3. Android Application

Development of the DRVC Android application was the most substantial component of the project. It used as a starting point the techniques learned through the Lab Assignments in GEOG 576, and added number of other features to support field collection of structure inventory data. In addition to the documentation that follows, a short demonstration of the application was posted to YouTube, **here**.

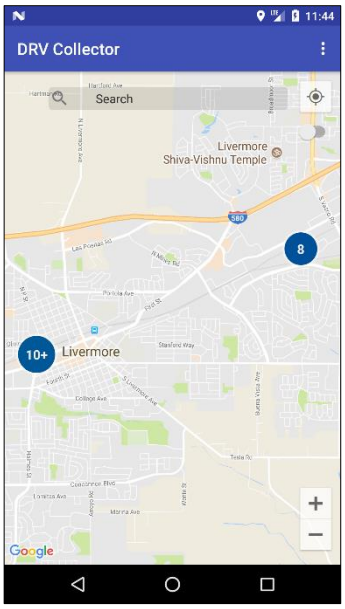
As a new programmer, the approach to development was to first envision how the app should function, and then research how to do that programmatically. To implement the features envisioned in the map required review and reliance on the Java and Android documentation available online, as well adaptation of solutions posted around the internet. Key resources which were adapted for DRVC are included in the bibliography and cited in the text.

The following subsections summarize the functionality provided by the application by discussing how the user interacts with the application, and then highlighting how that interaction occurs programmatically. To also provide some context for the project structures, the following classes are included in the project:

- MapsActivity
- EditRecordActivity
- OwnIconRenderer
- AsyncHttpPost
- RecordFilterActivity
- CreateRecordActivity
- MyItem

At a high level, the MapsActivity is the one that persists as the main activity, with instances of AsyncHttpPost and Create/Edit/Filter activities being created as needed. The MyItem and OwnIconRenderer classes were necessary to implement the marker cluster manger, discussed in more detail below.

Figure 2 – Start Screen



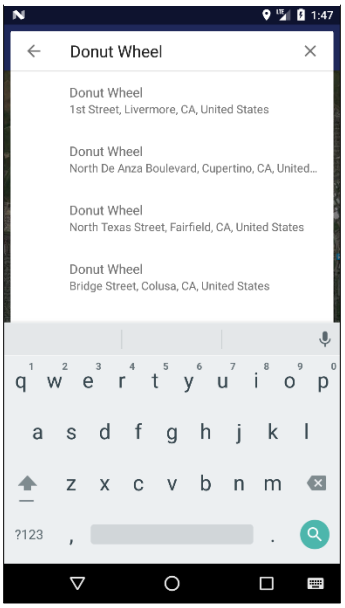
3.3.1. Main Map Screen User Interface

On launch, the user is shown a Google road map zoomed to an area that encompasses the records in the database. Clustering is used for aesthetics. Features available to the user include:

- onscreen zoom buttons
- "find my location" button
- toggle button to switch between road and hybrid base maps
- Google place autocomplete search box
- Dropdown menu with options to Filter Structures, Create Structure, Edit Structure, and Reset Map

The application includes permission checks and will prompt the user to allow use of location services and the camera. Google Maps API is used to retrieve the map. The Google Places API (8) supports the autocomplete search box. This activity utilizes a frame layout to allow positioning of elements anywhere on the screen, such as the map type toggle (9). Search box styling was customized remain visible over the street map and the typically darker aerial (10; 11).

Figure 3 – Google Place Search

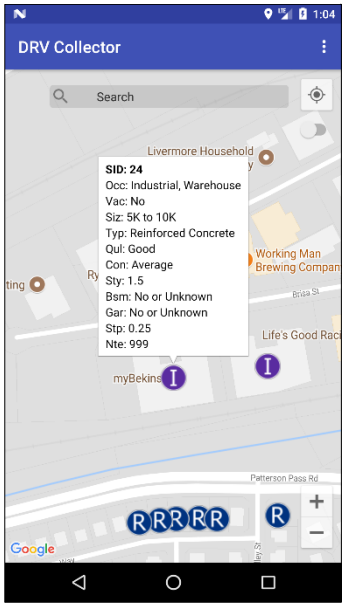


Behind the scenes, the map activity handles callbacks from the Create, Edit, and Filter activities, and pushes Post data to the class which performs asynchronous *doPost()* calls to the Java servlet. The maps activity also maintains several static helper variables to store intent request codes, file name variables, camera positions (12; 13), and search or click derived marker locations, all of which support some of the user interaction design decisions that were made. For example, most actions taken by the users which leave the map screen will return the user to same the camera position upon reloading the map, which was found to provide a more intuitive experience during field testing.

Lastly, the maps activity creates and uses on-click listeners, and handles related actions differently for markers not included in the cluster manager. These markers include those created from the auto-complete search box, or markers that the user can create by long-clicking on the map (14). Once a structure is loaded into the database, its marker will be handled by the cluster manager and will behave differently, as discussed below.

Markers for structures in the database are handled by the cluster manager and the *MyItem* class (15; 16; 17). These markers are generated in the *AsyncHttpPost* class when a result is received from the servlet. Custom icons are applied to markers representing structures in the database, with different icons based on the structure's category (residential, commercial, industrial, public, agricultural, or unspecified). A custom infowindow is also applied to these markers, and clicking on the infowindow for a marker in the database will initialize the device's camera, name the resulting image file with the structures' ID, and return the user to the app. Throughout the application, useful information is provided in toasts. In this case, the user is shown a toast with the filename of the photo.

Figure 4 – Custom Infowindow

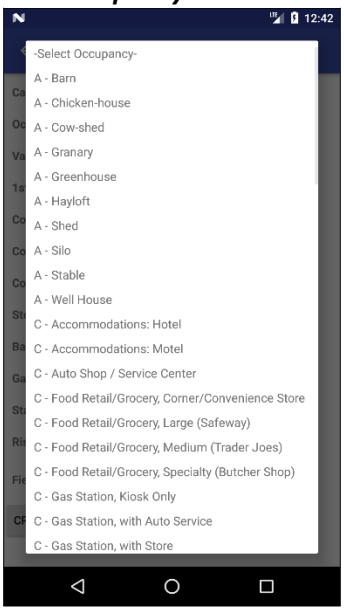


3.3.2. Filtering Structures in the Map

DRVC supports filtering the visible structures on the map according to the three structure characteristics are mostly like to be useful to users: category, occupancy, and construction type. The user is provided a menu screen with dropdown boxes and may select a from just one or all three fields (selecting nothing simply returns all the database records). Similarly, the Reset Map menu option is an empty filter query. A usability feature was implemented to return the user to the same camera position as before when performing a Filter, but resetting the camera to encompass all markers when Reset Map is used.

The filter occurs programmatically by creating a SQL SELECT-FROM-WHERE statement for the database to use. This begins in the *ReportFilterActivity*, which writes the user's selections from the dropdowns to an intent that is returned to *MapsActivity*. Then *MapsActivity* converts the intent data into a new *AsyncHttpPost* object which executes the request by passing data to the servlet. The servlet performs some final reformatting and provides a SQL statement to the database utility, which passes this to the database for execution. The database provides a result base to the database utility, which is passed to the servlet, which is served back to the *AsyncHttpPost* class. Upon receipt, the *AsyncHttpPost* class updates the set of markers to be displayed to the user.

Figure 5 – Occupancy Selector



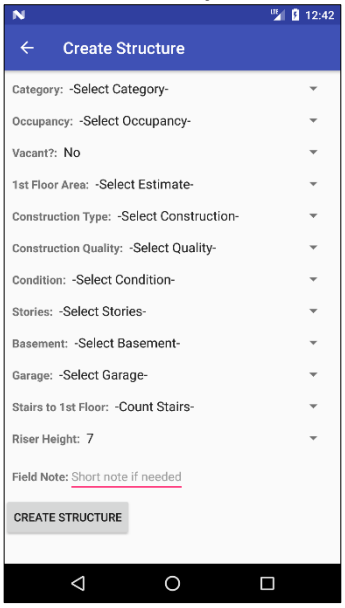
3.3.3. Creating New Structures in the Database

To make the application functional in the field, the user is provided three ways to create structures:

- Create Structure menu item: Created based on the device's GPS location.
- Auto-complete marker: Created at the location of the search result marker. The creation is started by long-clicking the marker's infowindow.
- Custom marker: Created at any location where the user has created a marker by long-clicking the map. The creation is started by long-clicking the marker's infowindow.

Programmatically, the application calls the same *CreateReportActivity* no matter which way the user initiates it. The only difference is where the latitude and longitude for the marker are sourced. The create structure screen includes a series of dropdown menus. Due to the number of selections for some fields, they are displayed at popout boxes rather than dropdown boxes. The occupancy field will show only relevant items if a category is selected first.

Figure 6 – Create Report



Execution of the report creation is very similar to the process described for queries; the main difference is what information must be passed down to the database. Upon successful creation of the new structure, the user is returned to the map showing the new structure.

The database and servlet are setup to allow mobile application users to specify only some of a structure's characteristics. When the user doesn't specify a characteristic, the fields will be passed a "NS" parameter for Not Specified.

3.3.4. Editing Existing Structures

Editing of existing records in the database has been implemented similar to the report creation action, but the user must also specify the ID of the record they would like to modify in the first row of the Edit Structure screen. The *EditReportActvity* also does not allow direct editing of geometry, which is handled instead by marker dragging (see Section 3.3.5). The user can initiate an edit by either long-clicking an existing structure's infowindow, or through the menu. Long-clicking passes the ID of the structure to the activity, but using the menu does not. The user

must then remember that ID value and launch the Edit Structure activity from the dropdown menu and then manually enter the ID. This ID is eventually used to create a SQL UPDATE statement. Future versions of the application will modify this workflow to populate spinners with existing values.

The custom infowindow adapter was used to be able to display many rows of data in the popup, as well as to be able to reposition the camera such that the marker is below center on the screen. As an extension of the editing function, a photo feature was also added to the application. Click on the infowindow for a structure will create a camera intent (18). The photo is saved to the device's internal storage in a /DRVC_images directory. Files are named by the convention "pic_SID_#.jpg." Like the Windows filesystem, the application will append "(#)" to the filename to avoid duplicates (19). DRVC does not currently include functionality to view photos associated with structures inside the application, and the none of the information about photos is returned to the database.

Figure 7 – Marker Dragging

3.3.5. Moving an Existing Structure

The main way that users can refine the location of structures in the database is by marker dragging. Dragging is implemented only for markers returned from the database, not for temporary markers generated from the search box or by long-clicking the map. Once the user releases the marker, the new location is sent to the database via the *EditReportActivity*, but only the structure ID and the new latitude and longitude are passed, which calls the MoveRecord method in the servlet. After completion, the database is queried for all records again and the map is refreshed, showing the marker at its new location, preserving camera position (20). Marker dragging is very useful in the field when locating structure points for large private parcels where the technician cannot gain access to the property, and must gather data from the sidewalk. In combination with an aerial basemap, marker dragging makes it possible to very accurately locate structure points that were pre-loaded into the database, such as the common practice of generating default points for every parcel centroid.

4. Results

Based on the DRVC prototype application and its development process, it is expected that DRVC could provide efficient and affordable alternative to typical commercially available tools. It is well-suited to the collection of field data as related to development of detailed structure inventory for natural hazard impact analysis. Even in prototype form, DRVC allows users to collect data quickly and consistently, which reduces the time required to cleanup data back in the office. The open source tools and the Google APIs used in the application are stable and well documented, and have a more than adequate feature set for collection of tabular field data. The three-tier architecture is simple, but robust enough to support heavy use of the application.

During testing of the application, several potential improvements and additional features were identified, and are described in Section 5.

5. Future Development

Several improvements and additional features will be investigated for inclusion in future version of DRVC. The following subsections document considerations related to each.

5.1. Improvements

5.1.1. Code Cleanup and Documentation

Due to the rapid development of the application, there are several places where the code can be improved.

- AsyncHttpPost class: This class currently includes too much functionality not related directly to interaction with the Java servlet. Most of the functionality related to cluster item listeners, clustering of markers, taking photos, marker dragging, and the custom infowindow for cluster items should be moved into one or more additional classes.
- The application's features are not documented well within the application itself. Toasts are used to display critical information, but a Help menu would be useful.

5.1.2. Structure Editing

The current workflow to edit existing structures could be improved. That a user may manually enter a structure ID introduces the possibility of data entry errors. It would also be useful to show a structure's current attributes in the Edit Report screen spinners, rather than only in the infowindows on the map.

5.2. Additional Features

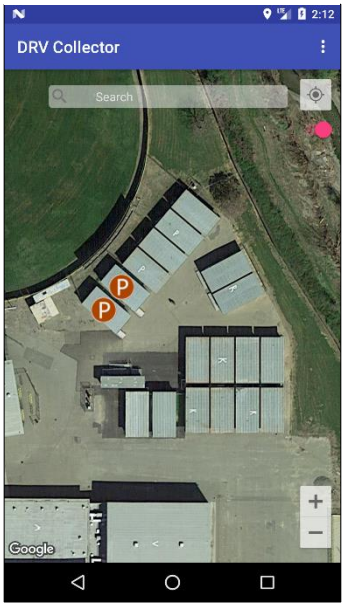
The following are additional features that may be implemented in future versions. The first three items discussed are the most critical.

5.2.1. Offline Mode

The ability to support offline editing is likely an essential feature if DRVC is to be deployed. Mapbox provides map caching capabilities, as well as a cache manager, and would likely be a good starting point for implementing this feature (21).

Disconnected editing of the structure database would also need to be addressed. One approach would be to have an offline mode which maintains a copy of the study area structures on the device. A "Structure" class could be created, and each structure would have methods for getting/setting the value of its fields. Then a "Sync" class could be created to setup a list of Create and Edit events that would need to be sent when network connectivity was again available.

Figure 8 – Copy/Paste Opportunity



5.2.2. User Interface Improvements

While performing field testing of the app, the most notable potential improvement aside from the Edit Structure workflow was the desire to copy an existing structure's attributes into a new structure. For example, when walking around a high school, the user comes across two dozen portable classrooms that are almost all identical. Given that each building requires selection of many characteristics, the user is suddenly faced with over a hundred clicks. Copy/Paste could save substantial time,

especially if the Edit Structure workflow is concurrently improved, which would make copy/paste useful anytime two structures are more alike than they are different.

5.2.3. Custom Map Layers

Inclusion of the ability to view custom geometry layers would be useful on projects where there are some existing datasets that need to be field-verified or provide useful information, such as parcel polygons. Implementation of this feature would require modification of all three layers of the DRVC application. GeoServer could provide suitable functionality for generating and serving WMS layers for geometry in the database (22). The Android Google Maps API also provides Tile Overlay utility which should be able to be pointed to the GeoServer (23).

5.2.4. Database Attachment Support

As noted in Section 3, photos may be taken and stored on the client device, but no photo-related data is sent back to the database. While the photos themselves may not be worth sending the database over the internet due to their size, it would be useful to record in the database whether a photo exists for a structure. Or better yet, the database could be modified to include a table for photo filenames (one-to-many structure) so that once back from the field, the user could batch copy photos to the database as attachments. Another photo-related feature might include geotagging of the photos themselves by editing the EXIF data on creation.

5.2.5. Complementary Desktop Tools

Because the DRVC application relies on open source tools and the flexible Google Maps API, the project could be extended to include a web interface which mimics the functionality available in the Android application. This interface would be useful for field technicians to use when back in the office or during the evenings when on field trips, as it would be more convenient to use the laptop for any data review and cleanup during these times. Another simple but useful feature would be to create a series of batch files to export the database tables to delimited text files which could be passed along to analysts performing the next phase of study.

5.2.6. Multiple Study Areas and Users

As currently implemented, all database entries are maintained in the same structure table. In the case where a user has multiple ongoing data collection efforts in different locations, resetting the map or running a query will select against all structures in the database. One approach would be to create an additional field in the structure table to store a Project Name code, and table of project names (one-to-many relationship), and provide an activity/screen for the user to select a project. After doing so, all SQL statements passed to the database would use that project id.

Another feature that could be added to the structure table is the name of the user recording the field data. This would be implemented like the project name, by storing a key in the structures table and adding a separate user table. The issue of concurrent users should also be considered if use of the application grows. In the current DRVC database, the default PostgreSQL transaction isolation level of “Read committed” was used. A stricter level may be more suitable if multiple users are committing data at the same time (24).

5.2.7. Security and Data Integrity

In this prototype, little consideration was given to data security or network security. Given the potential for breach via of SQL-injection type attacks, the application architecture would need to be safeguarded prior to deployment on networks with sensitive information. Additionally, while the DRVC application implements basic error handling and check to avoid null reference issues, these implementations could

be refined for efficiency and to provide more meaningful error and log outputs. Point-in-time recovery, via database backups and/or integrated history tables should also be considered.

6. References

1. **Melillo, Jerry M, Richmond, Terese TC and Yohe, Gary W.** *2014: Climate Change Impacts in the United States: The Third National Climate Assessment.* s.l. : U.S. Global Change Research Program, 2014.

2. **(ASCE) American Society of Civil Engineers.** 2017 Infrastructure Report Card. *2017 ASCE's 2017 Infrastructure Report Card.* [Online] (ASCE) American Society of Civil Engineers. [Cited: July 10, 2017.] www.infrastructurereportcard.org.

3. **(Corps) U.S. Army Corps of Engineers.** *IWR Report 95-R-9 Procedural Guidelines for Estimating Residential and Business Structure Value for Use in Flood Damage Estimations.* s.l. : Institude for Water Resources, Water Resources Support Center, U.S. Army Corps of Engineers, 1995.

4. **OpenSCG.** Postgres by BigSQL. *BigSQL.* [Online] [Cited: July 15, 2017.] https://www.openscg.com/bigsql/.

5. **The Apache Software Foundation.** Tomcat 7 Software Downloads. *Apache Tomcat.* [Online] [Cited: July 15, 2017.] https://tomcat.apache.org/download-70.cgi.

6. **JetBrains.** Free for students: Professional developer tools from JetBrains. *JetBrains.* [Online] [Cited: July 15, 2017.] https://www.jetbrains.com/student/.

7. **Vitalworks Internet Solutions.** No-IP Help. *No-IP.* [Online] [Cited: July 15, 2017.] http://www.noip.com/help.php.

8. **Google Developers.** Guides: Place AutoComplete. *Google Places API, Places API for Android.* [Online] [Cited: July 15, 2017.] Google Developers.

9. **Regmi, Pacific P.** Changing Map Types - Google Maps Android API Tutorial Part 4. *Viral Android.* [Online] [Cited: July 15, 2017.] http://www.viralandroid.com/2016/04/changing-map-type-google-maps-android-api-tutorial.html.

10. **Stack Overflow Forums.** Programmatically create textview background from drawable in android. *Stack Overflow Questions.* [Online] Stack Exchange Network. [Cited: July 15, 2017.] https://stackoverflow.com/questions/17759823/programmatically-create-textview-background-from-drawable-in-android.

11. **Android Developers.** Develop: Reference: Color. *Android Developers.* [Online] Google, Inc. [Cited: July 15, 2017.] https://developer.android.com/reference/android/graphics/Color.html.

12. **Stack Overflow Forums.** How to convert a LatLng and a radius to a LatLngBounds in Android Google Maps API v2? *Stack Overflow Questions.* [Online] Stack Exchange Network. [Cited: July 15, 2017.] https://stackoverflow.com/questions/15319431/how-to-convert-a-latlng-and-a-radius-to-a-latlngbounds-in-android-google-maps-ap.

13. **Google Developers.** Refernce: LatLngBounds. *Google APIs for Android.* [Online] [Cited: July 15, 2017.] https://developers.google.com/android/reference/com/google/android/gms/maps/model/LatLngBounds.

14. —. Guides: Markers. *Google Maps APIs: Android: Maps Android API.* [Online] [Cited: July 15, 2017.] https://developers.google.com/maps/documentation/android-api/marker.

15. **Stack Overflow Forums.** Android Maps Utils Clustering show InfoWindow. *Stack Overflow Questions.* [Online] Stack Exchange Network. [Cited: July 15, 2017.] https://stackoverflow.com/questions/30958224/android-maps-utils-clustering-show-infowindow.

16. **Shkurenko, Anton.** Work with ClusterManager. *Medium.* [Online] A Medium Corporation. [Cited: July 15, 2017.] https://medium.com/@tonyshkurenko/work-with-clustermanager-bdf3d70fb0fd.

17. **Google Developers.** Google Maps APIs: Android: Map Android API. *Google Maps Android Marker Clustering Utility.* [Online] [Cited: July 15, 2017.] https://developers.google.com/maps/documentation/android-api/utility/marker-clustering.

18. **Google, Inc.** Develop: FileProvider. *Android Developers.* [Online] Google, Inc. [Cited: July 15, 2017.] https://developer.android.com/reference/android/support/v4/content/FileProvider.html.

19. **Stack Overflow Forums.** Android/Java - When a file exists, rename the file. *Stack Overflow Questions.* [Online] [Cited: July 15, 2017.] https://stackoverflow.com/questions/16725499/android-java-when-a-file-exists-rename-the-file.

20. **Google Developers.** Changing Camera Position. *Google Maps APIs: Documentation.* [Online] [Cited: July 15, 2017.] https://developers.google.com/maps/documentation/android-api/views#changing_camera_position.

21. **Mapbox.** Offline Maps. *Mapbox.* [Online] [Cited: July 15, 2017.] https://www.mapbox.com/help/mobile-offline/.

22. **GeoServer.** GeoServer 2.12.x User Manual » Getting started » Publishing a PostGIS table. *GeoServer.* [Online] Open Source Geospatial Foundation. [Cited: July 15, 2017.] http://docs.geoserver.org/latest/en/user/gettingstarted/postgis-quickstart/index.html.

23. **Google Developers.** Google Maps APIs: Documentation. *Android: Anroid Maps API: Tile Overlay Utility.* [Online] [Cited: July 15, 2017.] https://developers.google.com/maps/documentation/android-api/tileoverlay.

24. **PostgreSQL.** PostgreSQL 9.1.24 Documentation. *PostgreSQL - The World's Most Advanced Open Source Database.* [Online] [Cited: July 15, 2017.] https://www.postgresql.org/docs/9.1/static/transaction-iso.html.