Northern Associates Incorporated

Core Logging Application

Project Final Documentation

UAA CSCE 470 Senior Capstone Project

Spring 2014

Keith Schneider
4/29/2014

Table of Contents

Abstract

The Core Logging Application is an Android based program that enables geologists to record data about core, RC and field samples. The application provides real-time quality control and data validation to ensure data integrity. The application was designed to integrate into existing data frameworks and provide to the collected data

Introduction

The NAI Geologic Logging Suite enables rapid logging of core, RC, and field samples. Providing near real-time dissemination of information through a local or global system allows for rapid decision making in the field or remotely based on the most current data available.

The ability to log straight to digital has demonstrated up to a 50% reduction in working hours, and reduced latency in the flow of information from data-collection to data-analysis in some cases by weeks to months. Geologic data collection lends itself to massive and complex data structures which the loggers are not typically aware of, this leads to time consuming and complex problems when transferring data from paper to digital. With real-time QC and recording data in a valid format from the start, digital logging saves frustration and man-hours for loggers, database administrators, analysts, and decision makers.

The Android application is part of a larger suite consist of three major components, an Android Tablet application, a Microsoft Excel component, and a master database. Only the tablet application will be considered for the purposes of this document.

The Android Data Collection Application

The Android tablet application allows loggers to collect geologic data and provides real-time QC. The straight to digital format enables data collected to be quickly checked by a database administrator, and to be put in the hands of decision makers.

Features

- Straight to digital Logging
- Real-time QC
- Integrated photos with annotation
- Bar-code scanning for laboratory samples
- Alternative interfaces for logging Core, RC, and Orientation
- The ability to transfer files in the field via USB, SD-Card, WiFi or Bluetooth

Description of Planning Process

The planning process for this iteration, involved consulting with my employers and clients. A list of requested changes and new features where categorized given priorities. One challenge faced in the planning process is that new requests for features or changes to existing data structures continue to come in, so the plan must be fluid and allow for additions and new priorities.

I used a system of nebulous units of time (NUTS) similar to the Agile method to assign approximate values to the individual user stories. The order in which user stories where implemented was based however on client priority.

Requirements

The original requested features list

Feature Requested	Description	NUTS	Category
Broadcast Master LUT	Enable Bluetooth Transfer of Files and Lookup Tables in App to all paired devices	3	Admin
Change LUTs In App	Ability to modify, change, or delete lookup tables.	2	Admin
Password Protect the Admin Interface	Create a password interface for administrators	2	Admin
Auto-generate the first record	Create the first record if non-exists, to prevent user frustration, or clearly indicate that no record exists.	1	Data
Collars Information	Allow for modification of the collars information table inside the application.	2	Data
Create User Level Documentation	Create Users Manual, port to HTML and provide links within the program that point to the correct document.	6	Documentation
Administrative Toggle For Sample QC	Toggle strict no-advance until right/QC on and off.	3	QC
Alt and Min Overlaps	Do not allow over laps	2	QC
Assign Sample ID	Ensure the user knows that the sample ID has been advanced, disable multiple clicks on the same record.	2	QC
Auto fill Toggle for sample depths	Toggle auto fill for sample depths. Delete footages for standards, and blanks upon selection.	1	QC
Background QC Thread	Runs Extensive QC routines that need to be off the main thread for responsiveness.	4	QC
Clasts QC	Clast Max/Average size QC	2	QC
Samples Sheet	Extensive and ruthless QC, on sample types and depths	3	QC
User Run Sample QC	Manually Run Record By Record QC for Samples	2	QC
Angular Measurement QC	Provide Notification if greater than the field value.	1	QC/Real-time

From and To Fields	If From > To, provide a clear and persistent indication.	1	QC/Real-time
Linked Percentage Fields	Provide Error notification on sum != 100%	1	QC/Real-time
Percentage Fields	Provide Error notification on > 100%	1	QC/Real-time
Rec/RQD Pct	Provide Error notification in > 110%	1	QC/Real-time
VS Point	Limit Decimal Places to 2/Real time QC on input values	2	QC/Real-time
Dirty Bit for Extensive Record QC	Use a dirty bit to indicate whether or not the record has been QC'd for process intensive QC items.	4	QC/Real-time?
Overlapping Records	Provide Notification/ (Potentially a Toggle) that forces correction of overlapping records.	3	QC/Real-time?
From and To in database	Only record metric, eliminate redundant fields. Use views/queries to record information for alternative applications.	3	Structural
HCL/Chemical	Add expandability for additional tests such as Kspar staining.	2	Structural
User Accounts	Create individual user profiles	3	Structural
Accentuate Field Colors	Create greater contrast in field colors, so they can be easily distinguished. Also move focus from text field if a pull down is selected.	3	UI
Action Icon	Make the Action Bar Icon return to the home screen	1	UI
Depth Auto Fill Toggle for all records	Provide a toggle that can turn on/off depth auto fill	1	UI
Digital Manuals	Links to HTML based navigable manuals	3	UI
From and To UI	Allow for input in desired units and perform a behind the scenes conversion keeping sig-figs. But Display only two sig-figs.	2	UI
Help Buttons	Use Help Buttons where appropriate	2	UI
Home Screen	Display Hole_ID and Loggers_Name. Provide a clearer interface that makes it explicit that the user must first click on the log which they want to open.	1	UI

List View QC Flag	Flag records to check with a visual indicator in the List View	2	UI
Log Creation Prompt	Adjust the log creation prompt to separate the first and last name field. Provide help content to assist in inputting proper user syntax. Hole ID Prompt PP-YY-###. Record User Initials for future use.	1	UI
Non-User Input fields	Make it clear, if it is not a user input field	1	UI
Opening Logs	Clearly Differentiate Between RC and Core Logs. Remove separate open log button, and have the computer determine the appropriate action.	1	UI
Orientation	Help Button with links to the manual	2	UI
recRQD Converter	Limit to appropriate digits of precision	1	UI
Split a record	Provide a long click option to split a record at a given depth. Providing identical copies with different from and to fields	3	UI
Struct Int	Default Cursor position to from field	0	UI
Swipeable Views	Spreadsheet View(editable?)/QC View/ Data-entry View	3	UI
Tab Order	Ensure Correct Tab Order on All Screens	1	UI
Alt and Min	Log/Level memory for which views are open	2	UI/Structural
Customizable Side View	Allow users to toggle on and off side view fields.	3	UI/Structural
HCL Depth	Add Depth Field and Mimic MagSusc	1	UI/Structural
MagSusc	Add flexibility for depth only and by interval	2	UI/Structural
MagSusc Device/Units/Comments	Add chooser for MagSusc Device/Units and Comment Field.	1	UI/Structural
Oxidation Screen	Style Above Intensity, Mimic Alt/Min	2	UI/Structural

Additional features added during the process

Feature Requested	Description	NUTS	Category
Quick Log Table	Create a new data sheet that includes a quick-log.	4	UI/Structural
Multiple Photo Support	Create a widget that allows multiple photos to be attached to a record. New Photo/Edit/Delete/View	3	UI
Certificate Based Licensing	Create an encrypted license system that is hardware specific.	3	DRM
Modify Fields and Calculations for Vein_Int	Add fields and modify calculations	2	Structural
Specific Gravity	Create a new datasheet to record multiple specific gravity measurements per record and additional QC functionality.	2	UI/Structural/QC

Description of Design

Controls

Controls are built upon a common interface that contains both the control component, and QC code that is particular to that type of control, or data field. This allows for the control itself and the code to vary independently and promotes reuse. Each control has a reference to it parents data structure and registers as an observer to be notified of specific changes in the structure.

Parent Views

The parent views consist of two major visual components. A navigational component which allows for navigation through the data structure. And a view consisting of the various controls necessary to make modifications to the actual data.

The parent view also contains data QC component that can check the data set as a whole for validation errors that can only be found in context. Validation errors can then be forwarded to the appropriate controls to notify the logger, that the data need further attention.

The Model

The database is based on the Open-Source SQLite3 format, and contains the recorded data. Queried data is accessed through a custom pointer class that navigates the data structure, performs updates to the data, and notifies registered listeners of changes. The model itself contains some inherent QC that enforces validation for critical information to prevent the dataset from being corrupted.

Software Development Process

The software is primarily implemented using a model view controller (MVC) paradigm. While specific functionality for controls are implemented as a component based method. This allows specific functionality to be implemented once, and then wrapped around a control.

When implementing new functionality such as a QC routine or a generic function, the widget base class is first extended. The base class already has logic built-in that knows how to interface with the data structure, and automatically registers to be notified of changes. The object then can listen and respond to either event based (user initiated) changes, or wait until notified by the data-structure.

UI elements are primarily implemented in XML, and are modified using a WYSIWYG editor, however within the application there are times when this is not feasible. In certain circumstances views or controls must be generated dynamically in code.

When structural changes to the database are required it has system-wide repercussions and must be implemented throughout the data suite. This requires involves implementing the imposed changes and testing the results in the application, and the rest of the system. Any time such a change is made all affected clients must immediately update all software used, so great care is taken with this kind of change.

Analysis, Results, Discussion

Overall the system that I have developed is both flexible and extensible. The modular bolt on component model for adding functionality to controls works well, and from a coding perspective minimizes redundant work. Implementing the UI in XML is a great benefit that comes along with the Android API, however for an application that is this large with a great many views it can be very time consuming. Presently all views are implemented for a 10.1" screen. However, it would be a huge undertaking to implement these views in a manner that would accommodate all form-factors, and would greatly increase the cost of any future modifications.

The programs inherent tie to the structure of the database is also a significant hurdle, where changes have serious implications. This is functional and economically viable on the scale that the application is deployed, but if this where to serve a significant number of clientele with unique needs, it would probably require a full-time dedicated support staff to maintain the application.

One proposed solution to both the UI and Structural issues would be to dynamically generate the required information at run-time using a scripting language. I actually accomplished this in a previous generation of the application, but it increases the code complexity, and discourages customized routines. However I would eventually like to come up with a data-descriptor that could be implemented at all levels of the application suite.

Conclusion and Lessons Learned

The application itself is field tested and proven, but will be constantly under development and updating for the lifetime of its usefulness. Every step I take to ensure maximum reuse of code, and general functionality has paid off in spades. There are certainly changes to the code base that could greatly extend the usefulness of the application such as employing a scripting language that defines the programs operation. The goal would be to maintain maximum client-centric customization, while generalizing the implementation. Previous attempts at this have shown that it comes at a cost, particularly in terms of runtime efficiency, and abandoning certain platform abstractions that provide convenient controls for styles and formatting.

Developing this application has impressed on me the importance of UI design, and especially in a resource constrained environment. Requests for structural changes to the application come almost exclusively from upper management, while request from the users are almost always UI centric. There is always a balance that has to be struck between the two, but in many ways the second is more important. If using the application becomes a frustrating experience, then its won't survive very long.