Memo

To: T. Ahlborn, D. Harris, L. Sutter, B. Shuchman, J. Burns

From: K.A. Endsley, M. Forster, C. Brooks

CC: D. Evans, R. Oats, K. Vaghefi, R. Hoensheid, de Melo e Silva

Date: March 30, 2011

Number: 17

Re: DSS development and decision criteria, beta version testing

---

The Decision Support System (DSS), a tool for helping bridge inspectors and managers to make informed and cost-efficient decisions about bridge maintenance, will be designed for flexibility with the multiple types of data generated by technologies evaluated in the course of this USDOT-RITA project. The relational database on the back-end is already capable of working with different datasets including geospatial data. We have already ingested a snapshot of the database used by MDOT in their bridge management system (BMS). This historical database of bridge condition information includes the following parameters that are accessible as decision criteria:

- Latitude
- Longitude
- Year built
- Year reconstructed (if any)
- Year painted (if any)
- Year overlaid (if any)
- Deck rating
- Substructure rating
- Superstructure rating
- Culvert rating (if any)
- Sufficiency rating
- Number of lanes
- Number of spans
- Material used in construction
- Paint type
- Deck width
- Bridge length

The DSS will incorporate multiple heuristics as needed to support bridge managers and other decision makers in evaluating bridge condition at an inventory level. A modular, web-based interface will allow the user to select only the decision from among different data presentations. Decision criteria will be presented one of four ways, or data “views” for the user: in a geographic information system (GIS), in an interactive chart, in a table, or in a custom "remote sensing view" for photographs and other metadata or a custom remote sensing data view (e.g. interpreted radar images).

Visualizing Bridge Condition Data in a GIS

The GIS view will allow for the categorization of various elements of bridge condition, displaying all bridges in an inventory symbolized or color-coded by such inputs as deck condition rating or date built. Its most basic contribution to decision support will be the identification of bridges based on their location—a “spatial query.” This will be enhanced by the symbolism of different bridge condition parameters or "attributes" at the inventory scale—in effect, the "color coding" of bridges on a map. More advanced visualizations can be derived from position and attribute information by contouring or gridding a large number of bridges. This will allow for more complex relationships to be compared in more than two dimensions; that is, geography and single attribute relationships captured by simply color-coding bridges on a map would be improved by the ability to compare geography and more than one attribute (e.g. color-coded bridges for one attribute overlain on a contour map of another attribute). Here are just a few examples of the kinds of decision criteria a GIS could present in this context:

- **Categorization**: Deck condition of all bridges in an inventory, region, or city
- **Generalization**: Contour or "heat maps" of year built (and, by extension, age of infrastructure)
- **Distance, cost or path analysis**: Minimum distance to another bridge of a given or similar condition, age, etc...
- **Spatial aggregation**: Number of bridges within a certain distance of a given point (e.g. MDOT regional office) or radius within a certain number of bridges matching the given criteria are found
- **Interpolation**: Metropolitan infrastructure with deck condition interpolated along roadways for dense, adjacent bridges

Visualizing Bridge Condition Data in Charts, Graphs, Plots

The chart view will provide the usual benefits that come from plotting data points—specifically, the visualization of the relationship between two or more parameters. In many cases, one of these parameters will be time. At a recent team meeting with MDOT (in Ann Arbor, on February 24), an MDOT bridge inspector expressed interest in viewing the deterioration rate for bridges. Deterioration rate plotted as a curve against time would represent a repeatable database query of interest—the kind of functionality that can be programmed into the DSS for repeat access. Trend analysis of a time series might not be very useful for this application as the trends are generally well known: bridge condition deteriorates over time, bridges get older over time, and maintenance of bridges at regular intervals improves their condition over time. However, curve fitting might allow for the estimation of deterioration rate at multiple scales. Decision criteria *at the inventory scale* that will be enabled by this presentation include:

- **Scatter plots**: Deck condition vs age, deck condition vs length, year built vs year reconstructed, paint type vs year painted, superstructure condition vs substructure condition...
- **Histograms**: Frequency of any bridge condition metrics
- **Pie charts**: Number of bridges given each rating for their deck, substructure, or superstructure; number of bridges in the inventory with virtually any parameter

For individual bridges, too, trend analysis of a time series is generally not as useful (as trend is known: bridges deteriorate over time). Again, the rate of bridge or structural element degradation might, however, be estimated from curve-fitting, an area investigated by MDOT Bridge Operations Engineer Dave Juntunen. Here are some decision criteria *at the scale of individual bridges* that will be enabled by this presentation:

- **Scatter/line plots**: Deck, superstructure, substructure or culvert condition of a bridge over time; plot of individual structural element ratings together
- **Trend analysis (if significant)**: Rate of bridge (whole) and bridge element deterioration
- **Multiple curve comparison**: Condition changes compared over time for bridge as whole; condition rating over time of every structural element
- **Pie charts**: Percentages of structural elements receiving various ratings throughout inspection history

Representing Bridge Condition Data in a Table or Grid

Despite the rich visualizations offered by the GIS and chart data views, a table of raw bridge metrics is still important for decision making as it offers insight into further relationships between data points and parameters that are not exposed the same way in a map or on a plot. Sorting, filtering, and aggregating records will be the most basic and essential functionality built into these data grids, offering insight into decision criteria that include the following:

- **Ranking**: Sorting records on any field finds the bridges built earliest, built most recently, in best/worst condition, with the most lanes or spans, etc.
- **Classification**: Multiple sorting of fields allows more complex classification such as the oldest bridges with the worst deck condition ratings or the bridges with the greatest number of both lanes and spans.
- **Exclusion/inclusion**: *Filtering* or querying records on multiple fields allows for very narrow categorization of bridges such as "bridges with a deck rating of less than 6 with no more than 2 lanes built before 1970 without paint type 3" to be found quickly.
- **Aggregation**: Fields can be summarized and totaled for any parameter such as the average deck condition rating of all bridges selected, the standard deviation of

sufficiency ratings, the most common year built or most common condition rating in an inventory...

Visualization of Bridge Condition Remote Sensing Data

The three views described so far make up the bulk of data presentation in a wide variety of fields for a number of applications. We have determined, however, that alone they are not quite sufficient for the representation of some datasets that are planned for incorporation in this study, namely remote sensing datasets. These data typically consist of imagery and other spatial representations in localized (i.e. individual bridge) coordinate systems. Here is a brief list of the kinds of data and their representation that should be enabled by the "remote sensing" view:

- **Radar image analysis**: Radar images of bridge decks, substructures or structural elements will be accessible in both their raw "blob" form and with advanced analysis techniques applied for feature extraction and/or defect identification; the latter results should be displayed on top of photos of the measured bridge, with annotations about detected cracks, delaminations, and their metrics; in many cases these overlays might be applied to a photograph of the bridge.
- **3D optical analysis**: 3D models derived from photogrammetry will be available in sa user-friendly form; the intended platform of the DSS (a web browser) is not currently a good environment for 3D modeling so it is likely that snapshots of 3D models or raster imagery of surface models will be made available instead, with the appropriate annotation about volume of spalls, depth of cracks, etc.

DSS Development: Moving from the Desktop to Mobile Devices

Our Decision Support System (DSS) will consist of a client-side interface developed in Javascript using the ExtJS library (http://www.sencha.com/products/extjs/). This library allows for the development of rich user interfaces in Javascript. When a website written with the ExtJS library is accessed from a browser, the HTML elements that make up the page, complete with dynamic behavior and event handlers, are rendered upon execution of the Javascript. As the library was designed for compatibility with all major desktop browsers on the market (e.g., Internet Explorer, Chrome, Safari, Firefox), it is during the execution of the Javascript code that the library dynamically responds to the client environment. Failsafe browser and OS detection is used when instantiating HTML elements and their associated behaviors so that only syntax compatible with that environment is used. The library exposes these detection schemes as a set of global Booleans (e.g. Ext.isIE6, Ext.isSafari) so that the developer can add additional, browser-dependent behavioral constraints (e.g. if (Ext.isIE) {// Code executed only for IE browsers}). Unfortunately, there is not Ext.isMobile to detect mobile browsers.

While the library is extremely reliable for desktop browsers, the library's performance on mobile browsers is not clear. For compatibility with the DSS, any candidate browser must be capable of:

- Understanding and executing Javascript
- Accessing Javascript files from external domains (e.g. Google Maps API)
- Performing GET and POST HTTP requests
- Receiving Javascript Object Notation (JSON) data in the body of an HTTP response

Additional features that are desirable, though not required, for candidate browsers include:

- Accepting and properly handling gzip-compressed (Javascript) files

From an independent developer's report (http://www.twintechs.com/blog?p=103), it appears that both browsers we tested here (iOS Safari and the built-in Android browser) support gzip compression.

Initial testing was conducted with our iOS (iPad/iPhone) and Android (Samsung Galaxy Tab) testbeds. With each device, we accessed web applications developed by MTRI using the same technologies that we plan to leverage for the DSS. These web applications are mostly (www.MichiganTechLakeSuperior.org/data) or entirely functional on mobile devices, though they obviously would benefit from design that was informed with a mobile browser in mind. Evaluation of the performance of these testbeds was largely based on our experience with the aforementioned web applications, as mobile devices do not offer the same developer's tools (e.g. Firebug) that desktop browsers do which would enable a more rigorous investigation. As it stands, our preliminary investigation reveals the following about critical feature support of mobile browsers:

	Safari on iOS	(Built-In Browser) Android
Understands and executes Javascript?	**Yes**	**Yes**
Access external Javascript?	**Yes**	**Yes**
Perform GET requests?	**Yes**	**Yes**
Perform POST requests?	**Yes**	**Yes?**
Receive JSON data?	**Yes**	**Yes**
Works with Highcharts library?	**Yes**	**No?**
Buttons, fields, drop-down menus work?	**Yes**	**Yes**
Scrolls overflow into view?	**No**	**No**

The lack of support for the Highcharts library on the Android testbed is not yet understood. There is no clear reason as to why one Javascript library works while another does not. The problem is likely one of available memory, instead, as the Android testbed seems to have initiated the process of drawing Highcharts charts (it displayed the "Loading..." waiting message) but never actually rendered a chart.

Beyond the initial critical features we identified, we discovered some additional user-interface (UI) constraints of mobile browsers in our investigation. With the Samsung Galaxy Tab in particular, which is much smaller than the iPad, screen size or "real estate" affects the functionality of UI components which were designed with the desktop in mind. Smaller buttons used to close or collapse windows were extremely difficult to click on using your finger on a touch-screen (Figure 1). In addition, regardless of screen size, the display of overflow (content that needs to be scrolled into view) is not as expected on mobile browsers. In fact, in most cases, the option to scroll hidden content into view was simply not available. This needs to be addressed when developing with the ExtJS library. It is obvious that a separate interface will be needed for mobile devices—one with a user-interface tailored for the touch-screen environment (e.g. bigger buttons, no need for scrolling).

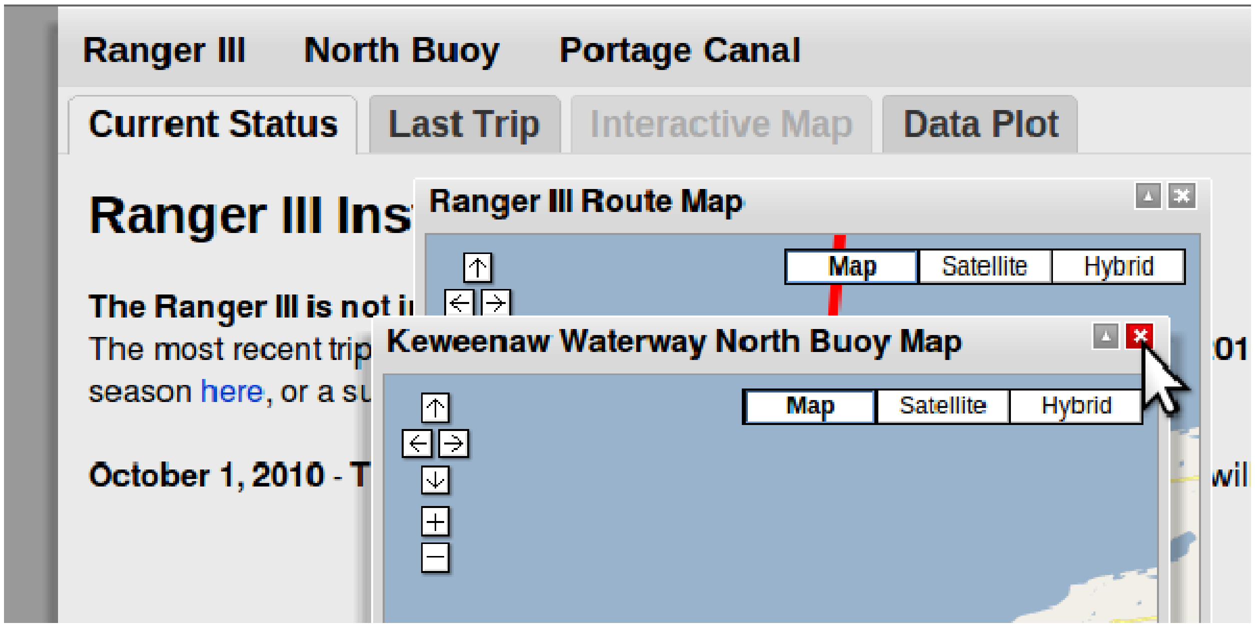

Figure 1: While the buttons to close or collapse windows are an appropriate size for desktop users, they are extremely small targets for a human finger on a mobile, touch-screen interface, as shown on this screenshot from www.MichiganTechLakeSuperior.org/data

While it is certainly an option to develop only one interface, intended for both the desktop and mobile client but with an interface optimized for mobile users, such an interface would hinder the desktop user's experience; in particular, it would be an inefficient use of screen real-estate. There are also other advantages to having separate desktop and mobile interfaces. Not only would they each cater to the strengths of their respective environments, but the overall size of the application (in bytes) would be reduced significantly compared to directing mobile and desktop users to the same website which must include the code for both interfaces. A desktop-focused interface would also work on smaller "netbook" type computers; our MDOT partners have recently experimented with making these available in their bridge engineer pickup trucks. We will investigate this platform in addition to the tablet devices and more traditional laptop or desktop computers.

It is important to note that developing two interfaces does not necessarily involve duplication of effort. In our opinion, the development workflow in ExtJS supports development of a desktop interface first. The real functionality and working components of the DSS will not be rewritten for the mobile interface. Rather, the user interface, once completed for the desktop, would then be optimized for a mobile environment. The changes required should not include functional (i.e. algorithm or data processing) changes. We anticipate that dimensions, position, appearance, and some minimal behavioral changes to the desktop interface are all that are required to produce a mobile-optimized interface.

In the event that the interface cannot be optimized for mobile devices using the ExtJS library due to unavoidable technical constraints (e.g. overflow scrolling), we have the option to develop a reliable, intuitive mobile interface using the Sencha Touch (http://www.sencha.com/products/touch/style-design/) library. As both ExtJS and Sencha Touch are created by the same developer and are both written in Javascript it is likely there will be minimal duplication of effort. Ideally, we wish to develop a desktop DSS in ExtJS and then modify it for mobile devices using the same library. Performance on mobile devices will be a consideration throughout desktop DSS development. Regardless of how they are developed, both the mobile and the desktop interfaces would provide, necessarily, the same functionality and capacity for decision support tailored for their respective operating environments.