LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-727102

# UI Review Results and NARAC Response

J. Fisher, B. Eme, S. Kim, K. Fischer, J. Donetti

March 17, 2017

## Disclaimer



This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

# Inter-Program Design Review of User Interface Frameworks for the NARAC Modernization Project

John Fisher, Bill Eme, Sei Jung Kim, Kathleen Fischer, John Donetti
National Atmospheric Release Advisory Center (NARAC), Lawrence Livermore National Laboratory

LLNL-TR-727102

*March 8th, 2017*

## Table of Contents



## Introduction

This report describes the results of an inter-program design review completed February 16th, 2017, during the second year of a FY16-FY18 NA-84 Technology Integration (TI) project to modernize the core software system used in DOE/NNSA's National Atmospheric Release Advisory Center (NARAC, narac.llnl.gov). This review focused on the graphical user interfaces (GUI) frameworks. Reviewers (described in Appendix 2) were selected from multiple areas of the LLNL Computation directorate, based on their expertise in GUI and Web technologies.

An FY14 DOE NA-42 (now NA-84) Technology Integration (TI) Scoping Study, and an independent Schubert committee project review, documented the urgent need to modernize the NARAC software system [Larsen et al., 2014a; Larsen et al., 2014b; Sugiyama et al., 2014; Schubert et al., 2014]. The detailed multi-year work plan to implement NARAC software modernization was favorably reviewed by the external Schubert review committee, made up of members from multiple agencies, and also internally in LLNL by non-NARAC personnel representing scientific, and computational disciplines in LLNL. As part of this modernization software architecture (see *Design of the Modernized NARAC Software Control Framework* [LLNL-TR-702854]), the GUI framework is going through a ground-up rewrite.

## Response to Committee

The specific comments provided by the inter-program design review committee are provided in Appendix 1. Below is NARAC's response:

- We agree that the significant changes from Angular 1 to Angular 2 was of concern. But, our conclusion is that Google has made a strong commitment to minimal breaking changes, with the revised APIs. For example, the next version (Angular 4) will be backward compatible with Angular 2 (https://www.genuitec.com/angular-4/). Time will tell of course, but for now, we feel our use of Angular 2 is an acceptable risk. Developers on the team who have used both Angular 1 and Angular 2 found the API changes to be a huge improvement. We've posted several StackOverflow questions, and immediately received answers back.

- After considering the review committees feedback, we've decided to NOT go with Electron, and instead go with a more traditional client/server model using the Express web server to server up web pages, and a RESTful interface. This has the following benefits:

    - We were continuing to encounter compatibility problems with Electron and Angular CLI, as well as Electron and node-java. These problems go away entirely when using Express.js as a service architecture.

    - By using a more traditional REST-based separation of client and server, we will better encapsulate our node-java bridge, which the committee (rightly) expressed concerns about. If node-java goes away, we can leverage a Java EE server (e.g. WildFly, Tomcat, etc) to provide the same REST APIs to our Java codebase, without impact to our client codebase.

    - We can take advantage of Websocket APIs to inform users of changes being made in the system, in real time. The ws library (https://www.npmjs.com/package/ws) appears to be a good option.

    - By using a more traditional client/server web solution, this will simplify the transition of technologies and capabilities from the Central System to the external Enterprise System.

    This approach does introduce some drawbacks that we had been avoiding with the Electron-based approach:

    - Additional dependence on a web server process that may introduce complexity or problems when running our GUIs on LC systems (if we wanted to provide that).

    - We will somehow need to allow users to bring up legacy GUIs, until all GUIs are converted over. Many of the GUIs are started in a particular "context" (for example, when using the Event Viewer, a user would "drill down" to the Report Generator GUI). With a browser-only interface, it's much harder to bring up the Java-based legacy GUIs directly from the Web GUIs. We're discussing possible solutions.

- We agree with the committee that we need to develop a testing infrastructure. As part of our move away from Electron, we employed Angular CLI (https://cli.angular.io/), which

automatically generates scaffolding for Karma (http://karma-runner.github.io) and Jasmine (https://jasmine.github.io/).

- We do have Continuous Integration (CI) and code review processes in place for the modernization effort, actively used by developers of the Java code. The user interface codebase will use the same processes very soon (once we get a bit more stable). The early code we have is going through ***extra*** code review at the moment, to make sure we have the code patterns right.

- We appreciate the recommendation of the geoxml3 package. When we explore a more complete visualization system using this angular/Typescript architecture we will certainly look into this package for our visualization system.

## Appendix 1: Feedback from Review Committee

The NARAC team presented a technology stack and prototype for the replacement of several NARAC tool user interfaces. Upgrading the UI is expected to be a multi-year task and represent a significant investment from the sponsor. Feedback from GS-CAD staff was sought. The proposed technology stack is most succinctly summarized as Angular2 with TypeScript in Electron utilizing a networked file system and possibly the Node-Java bridge. A prototype UI was briefly demonstrated. The prototype contained workflow features that appeared to address issues with the existing UI. Layout controls and general look and feel appeared modern and clean. Reviewers had several questions addressed during the meeting and the demonstration generated interaction and feedback.

NARAC developers presented the following justifications for their technology choices:

- Using modern web technologies will ease talent retention and acquisition.
- Electron allows the UI to remain a thick client while utilizing web technologies and code that are rapidly developing and largely adopted.
- A non-functional goal is to avoid all licensed technologies and maximize leveraging open source software.
- Critical technologies are supported or maintained by significant corporations. E.g. Angular2 by Google.

Several risk factors associated with the new design were identified.

- **Rapid evolution of JavaScript frameworks:**
  The new UI is all-in on Angular2. Angular is vastly popular, but fractured between incompatible versions 1 and 2. StackOverflow contains 220,000 questions on angularjs, but only 37,000 questions on angular2. Questions on angular2 are comparable to reactjs, a competing framework. Google has a recent history of deprecating and abandoning products. Google's commitment to maintaining backwards compatibility with Angular2 is understandably questionable.
- **Testing and testability:**
  Based on responses to questioning, testing and testability had not yet been investigated. Testable code is typically more well designed and cost-effective to maintain. Applicability of

testing frameworks and overall testability should be a critical element of planning before committing to the new technology. The CAPS team has begun using Jasmine.

- **Multi-user concurrency via network filesystem interactions:**
  The commitment to a thick client comes at the expense of potential concurrency enforcement through backend services. Utilizing a networked filesystem for concurrency and communication may prove challenging or ineffective. A distributed synchronization utility may still be needed.
- **Node-Java bridge:**
  The Node-Java project is still in beta releasing under version 0.*. The project describes several peculiarities and requires additional steps for Java 8 code. Relying on Node-Java in the design may be an unacceptable risk for a large multi-year redesign project.
- **UI usability and user metrics:**
  The prototype UI included the ability to customize layout and windowing of widgets. Workspace customization features offer power users extensive control of the application to maximize personal efficiency. To be effective the scientific staff may need training and workspace customizations may need to be persisted. A preliminary review with the application users may provide valuable guidance on the level of effort that should be invested. UI metrics tools can help provide continued usage insights.
- **The second-system effect:**
  Developing a second system may lead to feature creep and bloat in the new system while simultaneously abandonment of commitment to maintain the existing system. NARAC developers have lived with often painful issues in the existing system. Designing a new system may lead to a desire to add new features that are not critical.
- **Map APIs:**
  OpenLayers was prototyped as the chosen mapping API. Reviewers from the BKC have integrated NARAC generated KML into the BKMS application and chose OpenLayers. OpenLayers v2 and v3 do not support all KML tags. OpenLayers source had to be downloaded and modified to support GroundOverlay and LatLonBox. Several additional standard KML tags are not supported by OpenLayers. Google Maps API requires all KML be publicly accessible. The geoxml3 library for Google Maps allows rendering KML without making it publicly available, but will not work with the deprecated Google Earth Enterprise/Google Fusion maps on private networks.

The reviewers have the following general recommendations:

- Reconsider using a services architecture and serving the UI as a “thin client”
- Establish a CI process
- Investigate testing and establish test tools in a CI process
- Adopt a Git workflow model
- Utilize code review as a semi-formal process
- Identify language or component experts to maximize knowledge sharing in the code review process
- Avoid beta technologies that would not be easily replaced (e.g. Node-Java)

Links:

https://developers.google.com/maps/documentation/javascript/kml
https://github.com/geocodezip/geoxml3
https://github.com/openlayers/openlayers/issues/2941
https://github.com/joeferner/node-java
https://jasmine.github.io/

http://stackoverflow.com/questions/tagged/angularjs
http://stackoverflow.com/questions/tagged/angular2
http://stackoverflow.com/questions/tagged/reactjs

## Appendix 2: The Reviewers

The following people participated as inter-program, subject matter expert reviewers:

| Name | Current Project |
|---|---|
| **Sam Fries** | Analytics and Informatics Management Systems (AIMS) project, in the Climate Program |
| **Daniel Howell** | Biodefense Knowledge Center (BKC) Program |
| **Tim Bender** | Biodefense Knowledge Center (BKC) Program |
| **Kyle Dickerson** | Counterproliferation & Operational Intelligence Support (CPOIS) |
| **Eric Pernice** | National Ignition Facility (NIF) |
| **Ian Lee** | Livermore Computing (LC) |
| **Emily De Santis** | Global Security E Program Chief Engineer |

## Disclaimer and Auspices