# An Experimental Project Course to Prepare Students for Agile Web Application Development

Nicholas K. Clark
Dept of Computer Science
and C4I Center
MS4B5 George Mason University
Fairfax, VA 22030
+1-703-993-1743
nclark1@c4i.gmu.edu

J. Mark Pullen
Dept of Computer Science
and C4I Center
MS4B5 George Mason University
Fairfax, VA 22030
+1-703-993-1538
mpullen@c4i.gmu.edu

Christopher D. Bashioum
Defense Intelligence Information
Enterprise (DI2E) Framework
3076 Centreville Rd
Herndon, VA 20171
+1-703-464-9020
Chris.Bashioum@di2e.com

## ABSTRACT
Commercial software development today is dominated by the dramatic growth of Web/Internet-delivered information, combined with development techniques that feature rapid creation of reliable, maintainable application software. Staying current with these technologies is a constant challenge in any computer science curriculum. This paper reports on a pilot project course for six computer science fourth-year undergraduates that addressed this challenge in a synergistic way, to the benefit both of the participating students and of a government sponsor. The project created a new, open-source implementation of the Common Map API (CMAPI). It was accomplished using agile development techniques, intensive team activities, and advice from an industry CMAPI expert. Three one-month "sprints" resulted in a working, open-source system that met the needs of the sponsor and provided an excellent learning experience for all concerned. The approach used should be considered for inclusion in every undergraduate computer science curriculum.

## Categories and Subject Descriptors
D.2.2 [**Software**]: Software Engineering – *evolutionary prototyping.*

## General Terms
Design, Economics, Management, Reliability, Verification.

## Keywords
Open source, web application, agile development.

## 1. INTRODUCTION
Today's commercial software development is dominated by the dramatic growth of Web/Internet-delivered information, combined with Web application (webapp) development techniques that feature rapid creation of reliable, maintainable application software. Staying current with these technologies is a constant challenge in any computer science curriculum. Similar concerns also challenge government-sponsored software systems development, which has tended to fall behind the commercial world in recent years. This paper reports on a pilot project course for computer science fourth-year undergraduates that addressed both of these challenges in a synergistic way, to the benefit both of the participating students and of a government sponsor.

The sponsor in question, Defense Intelligence Information Enterprise (DI2E) Framework [6], is chartered to develop and maintain a common supporting information environment to be shared among networked software systems of the United States' military Services (Army, Navy, Air Force, Marines, and other related components). The DI2E Framework seeks to build on advantageous commercial and open-source software technologies and development techniques, providing timely capabilities to support those needs in the Defense Intelligence Environment.

As part of their effort to take advantage of the latest commercial technologies and development techniques, the DI2E Framework program office has supported student activities in plugfesting. Plugfests are events where rapid development/integration of networked software components is undertaken, based on open standards and generally also on open-source components. In this way they are able to explore software alternatives at relatively low cost while also nurturing a new generation of capable software developers.

The program office indicated an interest in sponsoring some of our students involved in plugfesting to undertake a related challenge, beyond the scope of a single plugfest: reimplementation of the Common Map API (CMAPI) outside of the Ozone Widget Framework (OWF) [16] where they had originally supported its development. The intended purpose was to demonstrate the broad utility of the CMAPI. They also agreed to provide the student project with an advisor having high expertise in both the CMAPI and contemporary software development methods.

The opportunity to implement the CMAPI as a student project was extremely attractive. The project would necessarily involve a team of several students, working collaboratively and employing the most appropriate commercial rapid development techniques as we understood them. It would be a challenging project that could draw on the technical strengths of our students while teaching them a great deal about methods for effective networked software development in a context broader than a typical one- or two-student senior project. Moreover, the sponsor was prepared to pay the students' course tuition plus a stipend and the project would produce published, open source software, which attracted the attention of some of our better and more employment-oriented students. This created a great educational opportunity that also met the sponsor's intention to demonstrate the broad utility of


*ITiCSE'15*, July 6-8, 2015, Vilnius, Lithuania.

CMAPI at a relatively modest development cost. We were able to recruit six students to undertake the project, which we cast as a one-semester project course executed in three "spirals" structured as "sprints" [21].

The first two authors of this paper are university faculty, who were responsible for conducting the project course; the third author served as the DI2E Framework advisor, formed an integrated team with the faculty members, and assisted in mentoring the students. The remainder of this paper is organized as follows: section 2 describes the Common Map API that formed the specification for the student project, section 3 describes how the project (and thus the course) was organized, and the next section describes how the three sprints were conducted. This is followed by a concluding section, describing how the CMAPI was demonstrated during the DI2E Plugfest, how the project course was carried out, and how it benefited the students while achieving the sponsor's goals.

# 2. COMMON MAP API

In today's online environment, a "map", or "web-map" is a graphic object that represents layers of geospatial data, metadata (e.g., URLs), or images in either two or three dimensions for human viewing and interaction [14]. The Common Map Application Program Interface (CMAPI) is a specification that codifies a standard way for data search/manipulation widgets (defined below) to be able to communicate with map widgets over a publish/subscribe (pub/sub) mechanism. The following text is paraphrased from the CMAPI specification itself [5]:

- Many programs and projects create webapps that search for or manipulate data, then present the results on a map. The CMAPI provides a standard that allows end-users or developers to combine data search/manipulation web apps from any vendor with map web apps from other vendors.
- The API is intended to work with any pub/sub based inter-widget communication mechanism. Messages are sent to the appropriate channels (as defined in [5]), and the map updates its state accordingly. Other widgets interested in knowing the current map state can subscribe to these messages as well.

## 2.1 Widgets and Web Applications

With HTML5 and the current web development mindset, the web browser can be thought of as the new application platform [22, 25], where windows or iFrames on a browser are similar to processes in an operating system (OS), and interwindow or interwidget communication within the browser is similar to interprocess communication (implemented by APIs) on an operating system.

The World-Wide Web Consortium (W3C) Mobile Web Application Best Practices Recommendation [23] defines a web application as "a Web page … or collection of Web pages delivered over HTTP which use server-side or client-side processing (e.g. JavaScript) to provide an "application-like" experience within a Web browser. Web applications are distinct from simple Web content … in that they include locally executable elements of interactivity and persistent state." The W3C Packaged Web Apps (Widgets) Recommendation [24] further defines widgets as "full-fledged client-side applications that are authored using Web standards such as [HTML] and packaged for distribution. They are typically downloaded and installed on a client machine or device where they run as stand-alone applications, but they can also be embedded into Web pages and run in a Web browser. Examples range from simple clocks, stock tickers, news casters, games and weather forecasters, to complex applications that pull data from multiple sources to be "mashed-up" and presented to a user in some interesting and useful way." In other words, a widget is a "packaged" webapp. A map webapp or widget is one that is designed to render both map images, and features as layers on top of the map images.

According to the World Wide Web Consortium (W3C) Web App Working Group: "The word *widget* is used to mean many things … A W3C Widget is specifically a packaged web application of any degree of complexity" [26]. Widgets often are implemented using HTML, CSS, and JavaScript or ECMAScript. According to the W3C Rich Web Client working Group: "Development of modern Web applications … depends on Web browsers supporting interaction between contexts, similar … to supporting inter-process communication in operating systems" [8]. In other words, communications among widgets.

## 2.2 Ozone Widget Framework

Four "patterns" of interwidget communication exist [27]: 1) message oriented, 2) shared memory, 3) Remote Procedure Call (RPC), and 4) pub/sub. There are also two "scopes" of interwidget communications: 1) single user/single browser, 2) multi-user/multi-browser. The Ozone Widget Framework (OWF) is a JavaScript based set of tools that implements the pub/sub pattern and single user/single browser scope of interwidget communications, and describes itself as "a web application for composing other lightweight web applications called 'widgets'. It is basically an extensive web portal engine, with the unusual characteristic that the content within the portal (i.e. the widgets) is decentralized. It includes a secure, in-browser, publish-subscribe eventing system, allowing widgets from different domains to share information. The combination of decentralized content and in-browser messaging makes OWF particularly suited for large distributed enterprises with legacy stovepipes that need to combine capability 'at the glass'." [16]

## 2.3 Project Purpose

Although DI2E managers believed that CMAPI could be applied in whatever technology implemented any of the communications patterns or scopes identified above, prior to the course described here the only extant CMAPI implementations required the OZONE Widget Framework (OWF) inter-widget communication platform technology. The sponsor's problem was to demonstrate CMAPI's applicability to various current platform technologies by implementing it in all four patterns and two scopes. However, due to time constraints, it was determined that implementing it using standard JavaScript and HTML5 cross-domain messaging technology (pub/sub pattern) with the two scopes would be sufficient to provide the required demonstration.

# 3. PROJECT AND TEAM ORGANIZATION

This course was unusual in that it also was a sponsored project run by our C4I Center [4], which undertakes research projects related to military information technology. It was structured to meet on a weekday evening like our other fourth-year undergraduate courses but the students were paid as if they were working ten hours per week in one of our laboratories, an arrangement normally considered only for graduate students but highly motivational to our undergraduates.

We recruited six students by means of email, sent to all potentially interested students we could identify. The students followed a less formal, hybrid development methodology that borrowed from

agile development techniques commonly found in Extreme Programming/XP [2], Scrum [21], Crystal [3], and Dynamic systems development method (DSDM) [20]. The students were organized into three teams of two. Based on a student suggestion, team members were rotated after each one-month sprint so each student experienced working with three others. The whole group, including faculty mentors, met weekly and was joined by our DI2E-provided advisor once or twice per month. Each sprint was concluded by a session where the advisor reviewed the running code presented by each team. This was consistent with best industry practices [21] and proved highly effective, in that students produced a working, open-source system posted on the open source repository GitHub [9] at the end of the third sprint. They named the system uMap (Universal Map). Heavily based on other open-source components, the resulting system has an innovative design and complexity well beyond that which could have been expected from individual students, especially in such a short timeframe.

To facilitate project management and coordination, the students used a set of open source collaboration tools. These tools made communication and coordination easier since most of the work was done in pairs working at different locations. In addition to scheduled weekly meetings, students met outside of class to work independently and all other coordination was done virtually. Collaboration tools used included:

- *Redmine:* A flexible project management webapp that includes project wikis, discussion forums, issue tracking, time tracking, and integration with source control management systems
- *Git:* A distributed version control system
- *Jenkins:* A continuous integration tool, Implemented for the third sprint
- *Google Docs:* Although not open source, the free 'shared-document editing service' provided by Google was used extensively for collaborating on documents, demo plans and presentations

# 4. DESIGN AND CODING SPRINTS

The sponsor did not provide a formal requirements document. The primary functional requirement was simply to implement the CMAPI outside of the OWF environment and do so using some form of inter-process/widget communication with the capability for single user or multi-user interaction.

## 4.1 Sprint 1: System Organization

Sprint 1 objectives:

- Understand and refine the requirements
- Explore alternative widget frameworks
- Explore inter-process/widget communication paradigms
- Define a system design concept by using exploratory prototypes
- Define components to use in Sprint 2

Students were advised that they should use any open source or commercial off-the-shelf (COTS) options available to minimize development effort and maximize simplicity and reusability. Each student pair selected an existing alternative widget framework to evaluate, demonstrate and critique. These included web-based framework/APIs jQuery [13] and the Dojo Toolkit [7] as well as a native application framework using GTK+ [11]. The discussions led students to agree on using a web-based approach with web browsers as the clients rather than native applications. They agreed that this approach would be simpler to develop and was instantly more portable to different platforms. Feedback from the industry advisor supported this approach with a suggestion of using web based clients with widgets in the form of iFrame objects running within browsers.

The students proceeded to evaluate communications paradigms for use between components with each student pair evaluating implementations of one of RPC, Message Passing (including pub/sub), and Shared Memory. These discussions produced two artifacts to facilitate the spiral 1 designs. The first was a common vocabulary of the system components as shown in Figure 1. The second was a set of criteria for evaluating the characteristics of communication components based on: real time 'across browser' support (message subscribe), ability to handle intermittent connections/disconnections, existence of at least one robust implementation, availability of multiple implementations, ease of use in a virtualized environment, support for multiple web browsers, ease of binding to a data storage provider, and having an open source license other than GPL.

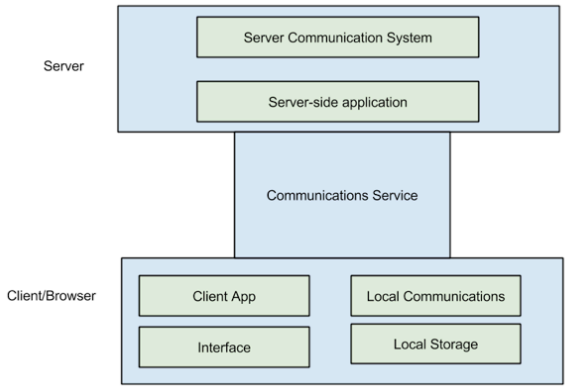


**Figure 1. Common Vocabulary**

The final product for each of the three teams in Sprint 1 was a system design with prototype demonstration. Teams One and Two produced fundamentally similar designs that only differed in the choice of persistent storage structures and specific Javascript libraries for communicating message traffic. They both included the capability for multiple widgets in a single browser instance as well as multiple user/browser connections. Both also required a local publish/subscribe capability within the single browser in addition to a library for client to server communication. Team Three's design abandoned the notion of multiple widgets within a single browser window and focused on multiple users in separate browser instances. Team Three also used a direct connection between each browser instance and a message queue server which utilized an in memory data storage server for persistence. All teams limited the scope of their initial prototype to implementing one or two functions of the CMAPI as a proof of concept. There was no formal code review of the initial prototypes since the objective was to use "throw-away" code to identify components to be used in the system design.

The group as a whole learned several important lessons from evaluating each of the three teams' designs. From Team Three, we learned that using direct connections from the browser to the queuing server in a wire-level format was overly complex in implementation. The group agreed the approach using web sockets through an intermediate web server as presented by the other teams was more efficient. However, Team Three's use of the in-memory data storage provided by Redis [19] was viewed by all as a more efficient solution for server side storage and solved

issues with storage performance identified by other groups. All of the groups also identified some aspects of the CMAPI that were vague to implement and would require input from the industry expert. From an educational view, the teams working on competing system designs proved a useful way to explore design ideas and encourage involved discussion.

The system design that went into Sprint 2 is shown in Figure 2. It was composed of components from among the different teams' designs. The students elected to include support for multiple widgets in a single browser as well as multiple user/browser support in their architecture. The client-side design included a Map Abstraction layer that enabled publish-subscribe type API calls to underlying map implementations such as Google Maps, OpenLayers, Leaflet, or ESRI's ArcGIS. The content displayed on the map was to be stored locally and managed by a Content Manager (CM), which would store and track the data referenced from the CMAPI messages such as overlayIDs and featureIDs. Other widgets could communicate with the map and content manager using the CMAPI messages through an Open Source Javascript library called PubSubJS. The server side used Node.js to serve the static content and used web sockets to both receive and transmit the CMAPI messages, which were also stored in Redis. REST services were used to manage access control.

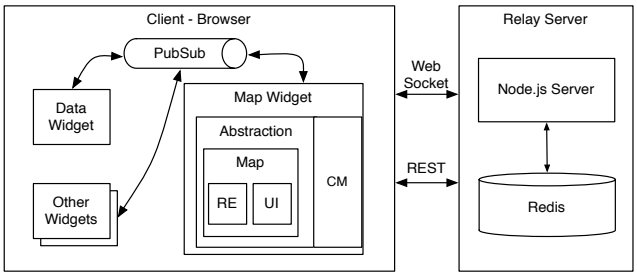


**Figure 2. Sprint 1 System Design**

## 4.2 Sprint 2: Solving the Hard Problems

The purpose of Sprint 2 was to identify the critical sub-components of the system design from Sprint 1 and build them into a working prototype. Sprint 2 was the most ambitious in terms of the amount of work needed to meet the objectives as it involved identifying and solving the harder, lower-level design problems. Some schedule and priority changes were necessary mid-sprint to address challenges encountered.

Sprint 2 objectives:

- Implement client-side code such as Content Manager for local storage, abstraction layer for Leaflet and ESRI, core set of map functions from CMAPI, and extending PubSubJS to provide publisher and source information
- Implement server code for inter-browser sharing of data leveraging Node.JS and Redis, including web server implementation for all REST calls to manage access to virtual rooms, and web sockets for pub/sub calls and JSON format for the data,
- Create and maintain documentation artifacts for design decisions and user documentation
- Build test cases and test harnesses

For Sprint 2, teams were no longer building competing system designs. Instead, each team focused on one part of the system. They naturally divided work into server and client components. With most of the complexity present in the client side, two teams focused there while one team worked on server side components.

On the client side, the map abstraction layer required an underlying geospatial component that renders geospatial information passed via the CMAPI into a visual map representation. The geospatial information referenced in the CMAPI can include a variety of types of information including points, lines, polygons, and images inside of container overlay and feature objects. It was not required that the students build their own map rendering system; they explored open source implementations such as Leaflet [15], Google Maps [10], and also the commercial ESRI ArcGIS for Javascript [1]. One team focused on the map implementation using Leaflet and ESRI in parallel. Leaflet was selected because it was designed for simplicity, ease of use, and performance, while ESRI's ArcGIS option was attractive as a COTS product.

During development, the map team discovered that the complexity of implementing two separate underlying map renderer libraries was greater than expected. They also encountered difficulty in implementing the Leaflet map renderer within the abstraction layer. They found that Leaflet's simplicity was actually a drawback in that it abstracted too much of the required functionality and thus would have required more development work. The group re-evaluated the available options and chose to focus on the Google Maps API for Sprint 2 because of its superior set of available references and examples, and to leave the ESRI implementation to Sprint 3. The group also pushed the Content Manager development into Sprint 3 and kept the local content storage directly within the abstraction layer.

Coordination of progress and scheduling became a challenge for the group during Sprint 2. Multiple discussion forum threads about development of different components were ongoing, but some students had trouble in conveying their progress and did not adopt a clear timetable for completion. Also, some students were so focused on their specific development that they had not considered the effort required for integration and testing with the overall system. The schedule for the Sprint 2 demonstration with the industry expert needed to be extended to allow time to address these issues. To address future coordination issues, one team leader was assigned to monitor and coordinate.

Although a delay of one week was needed to complete integration and testing, the final demonstration for Sprint 2 showed a working prototype to our CMAPI industry expert. The demo included both single user interaction within a single browser and multi-user/browser support leveraging Google Maps API version 3. All of the critical components were implemented for this demo. During this Sprint, the system design as shown in Figure 3 was altered to accommodate challenges with implementing decisions made in Sprint 1: the Content Manager was integrated directly into the map abstraction layer, and PubSubJS was replaced with another open source project called Porthole to support intra-browser communication among widgets and overcome issues with browser security on cross-domain communication [18].

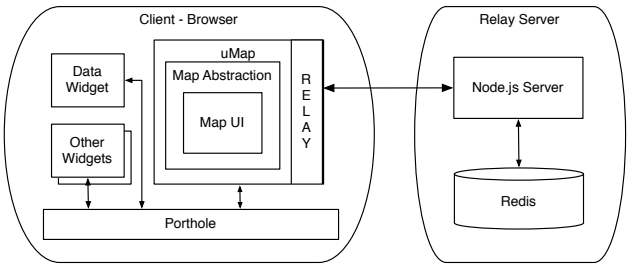


**Figure 3. Sprint 2 Design as Implemented**

## 4.3 Sprint 3: Working uMap System

Sprint 2 succeeded in building the critical components; however, there was still considerable work to build a polished product. The overall goal of the project, to be completed at the end of Sprint 3, was a demonstration of the complete system with the project sponsor.

Sprint 3 Objectives:

- Full support of the CMAPI with Google Maps renderer
- Support ESRI ArcGIS Javascript map renderer
- Fully integrate components
- Continuous integration testing environment
- OWF interoperation

Continuous integration development environments are an industry practice in which active developers merge new code with the main development branch at regular (and short) intervals [8]. Once the new code is merged, regression tests are performed, often nightly and through an automated system that notifies developers when a test fails. This approach is beneficial because it can catch problems early, reduce the amount of time tracing the change that caused the error, and reduce overall development time.

In Sprint 3, the student groups agreed that testing and continuous integration was an important early priority. A continuous integration build environment was created using the open source software, Jenkins [12]. Since a lot of the testing was based on client user interface interaction that would normally take place in a web browser, PhantomJS (a scriptable headless browser testing API that allows for fully automated tests without an actual web browser or user [17]) was used to facilitate testing. Image captures of rendered artifacts were automatically compared to saved images of the expected output, and when an inconsistency between the two was found during nightly testing, the developers were notified of the failure by email.

Aided by the testing environment, development time was reduced and the components were successfully integrated. This included a new room manager interface on the client that utilized the already implemented REST access control interface on the server, which enabled users to identify themselves and create, join, or leave a virtual room. When joined to a room, the client will send and receive CMAPI messages only within that room, allowing multiple users to carry on virtual geospatial chats (i.e., editing and manipulating shared maps) in different rooms simultaneously.

Demonstrating the robustness of the CMAPI was an important goal for this project. To that end, it was important to demo CMAPI features rendered on more than one underlying map visualization product. One team member focused on this implementation and in the final product we had the capability to plot features onto both Google Maps version 3 and ESRI ArcGIS maps. However, the ESRI implementation was completed only at proof of concept level; uMap used Google Maps for the full multi-user demonstration.

uMap's conformity with the CMAPI specification was also demonstrated via interoperability with some extant CMAPI widgets within the OWF environment. This was accomplished by creating a custom OWF-based "uMap Relay Widget." This widget relayed CMAPI messages between OWF based map widgets and the uMap server. Thus, when users using the uMap browser client would manipulate geospatial information on their maps, the OWF users could see the same information and vice versa.

At the end of Sprint 3, the teams disbanded and all six students worked extensively on the demonstration script and plan for a final presentation to our sponsors at the DI2E Framework. Their hard work paid off in a successful demonstration of uMap that included extensive documentation, Google and ESRI map implementations, multi-browser virtual room support, and interoperability with all existing OWF CMAPI Widgets. The project sponsors were pleased with the success of the project both in representing the robustness of the CMAPI standard and identifying areas that were difficult to implement outside of OWF, such as the scheme used to generate unique identifiers for overlays and features. They also expressed interest in further development of some features that were not included in the Sprint 3 design such as the ability for late joiners to a room to catch up with previously transmitted information.

The final product, uMap, is an open source implementation of the CMAPI. The uMap framework of tools provides real-time network collaboration between operators, which OWF does not. uMap includes a mapping provider widget which supports Google Maps and ESRI. Other mapping providers are possible. The map provider natively uses the network relay tools to provide the network collaboration features such as editing maps with other users in a virtual room. It supports all applicable features in the CMAPI 1.1 standard. A uMap example interface is shown below in Figure 5. The uMap code is available on GitHub at https://github.com/gmuc4i/umap. An example implementation is available at http://styx.c4i.gmu.edu:3000/.

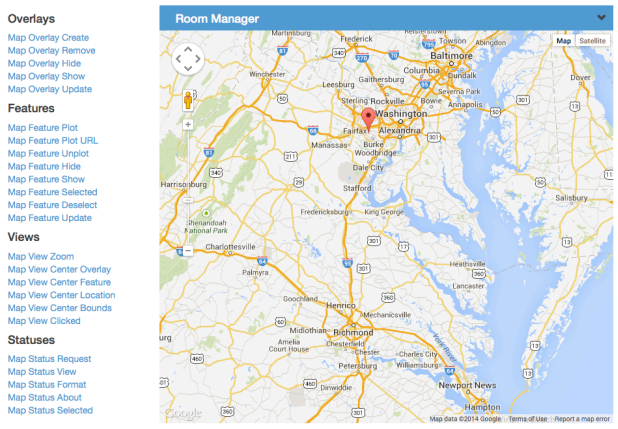


**Figure 5. uMap Example User Interface**

# 5. CONCLUSIONS/FUTURE WORK

The project met its multiple goals: students learned a great deal about webapp programming and agile/team development; faculty learned effective ways of teaching these things; and the sponsor achieved the sought confirmation of the generality of CMAPI along with some caveats about the same.

Sprint 3 of the uMap project was completed two weeks before the DI2E Plugfest 2014, a major event that was hosted at our university. The sponsor encouraged our students to participate in the event and demonstrate uMap. The DI2E Plugfest includes a subevent called the "Mashup Challenge" where government-sponsored integration teams integrate the interoperable applications provided by industry participants and made available through the "DI2E Storefront," a repository of such applications. The students built a mashup (an interoperating group of webapps) around the uMap. Their mashup involved a scenario where intelligence analysts collaborate using uMap to track the

movement of a suspected terrorist and an illegal arms shipment. They leveraged several data query services present in the DI2E Storefront and visually collaborated using different map products through uMap. The result drew considerable positive attention by government and industry attendees at the plugfest event.

Web programming and software development by agile teams both have emerged as important areas for mastery by computer science students. The project described in this paper demonstrated conclusively a creative approach to introducing these current topics into the upper-division curriculum. Our students grasped the new approaches eagerly and performed admirably. The ability to also meet a current business need and provide financial support to students while exploring this approach were additional benefits that will not necessarily be available to other faculty who expand the computer science curriculum into these areas. However, we believe that one key aspect of our work is essential to the computer science curriculum: the state of practice in software development has come to a point where effective mastery at the baccalaureate level requires that student participate in small, well-structured teams solving current real-world problems. We therefore intend to develop a regular fourth-year elective project course similar to the one described above, but with a requirement for local industry guidance rather than a supporting sponsor. We encourage computer science faculty members from other institutions to consider doing the same.

Toward this end, we offer these organizing principles for an agile webapp development project course:

1. Begin with worthwhile, well-defined problem set and knowledgeable representative user or surrogate user
2. Build on available open-source and COTS, in Web environment
3. Insist on a small team (4 to 8) per product
4. Employ short, high-intensity sprints (3 or 4 in semester) producing documented, running code products presented to engaged audience at end of sprint
5. Within teams, have students work in pairs (or maybe triples) that rotate for each sprint
6. Define sprint products at beginning of each sprint and monitor progress in weekly team meetings with faculty
7. Expect success but watch for stumbles and involve team in immediate remediation

## 6. ACKNOWLEDGMENTS

Our thanks to DI2E Framework Program Office for its support.

## 7. REFERENCES

[1] ArcGIS API for Javascript. Retrieved December 31, 2014 from https://developers.arcgis.com/javascript/jsapi/

[2] Beck, K., Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999

[3] Cockburn, A. Crystal clear: a human-powered methodology for small teams. Addison- Wesley Professional, 2005

[4] Center of Excellence in Command, Control, Communications and Computing, Retrieved December 23, 2015 from http://c4i.gmu.edu

[5] Common Map API Specification. Retrieved December 26, 2014 from http://www.cmap.org

[6] Defense Intelligence Information Enterprise (DI2E). Retrieved December 23, 2014 from http://www.dtic.mil/ndia/2014system/16835WedTrack6Azizian.pdf

[7] Dojo Toolkit. Retrieved December 23, 2014 from http://dojotoolkit.org

[8] Duvall, P., S. Matyas, and A. Glover. Continuous Integration: Improving Software Quality and Reducing Risk. Pearson Education, 2007.

[9] GitHub, Inc. open source repository. Retrieved December 23, 2014 from https://github.com

[10] Google Maps API. Retrieved December 25, 2014 from https://developers.google.com/maps/

[11] GTK+, The GIMP Toolkit. Retrieved 23 December 2014 from http://dojotoolkit.org

[12] Jenkins. An open source continuous integration server. Retrieved December 23, 2014 from http://jenkins-ci.org/

[13] jQuery, Retrieved December 23, 2014 from http://jquery.com

[14] Kraak, M. & A. Brown (eds). Website accompanying Web Cartography: developments and prospects. Retrieved December 31, 2014 from http://kartoweb.itc.nl/webcartography/webbook

[15] OpenLayers. Retrieved December 25, 2014 from http://openlayers.org

[16] Ozone Widget Framework, Retrieved 26 December 2014 from https://github.com/ozoneplatform/owf

[17] PhantomJS. Headless scriptable WebKit with JavaScript API. Retrieved December 26, 2014 from http://phantomjs.org

[18] Porthole. JavaScript Library for Secure Cross Domain iFrame Communication. Retrieved 25 December 2014 from http://ternarylabs.github.io/porthole/

[19] Redis. Retrieved December 23, 2014 from http://redis.io

[20] Stapleton, J. DSDM: The Method in Practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1997

[21] Sutherland, J. *Scrum: the Art of Doing Twice the Work in Half the Time*, Crown Business, New York, 2014

[22] W3C, Browser Technologies, Retrieved 31 December, 2014 from http://www.w3.org/wiki/BrowserTechnologies

[23] W3C, Mobile Web Applications Best Practices, W3C Recommendation 14 December 2010. Retrieved December 31, 2014 from http://www.w3.org/TR/2010/REC-mwabp-20101214/

[24] W3C, Packaged Web Apps (Widgets)-Packaging and XML Configuration (Second Edition), W3C Recommendation 27 November 2012. Retrieved December 31, 2014 from http://www.w3.org/TR/2012/REC-widgets-20121127/

[25] W3C, Rich Web Client Activity. Retrieved December 26, 2014 from http://www.w3.org/2006/rwc/Activity

[26] W3C, Widget/Webapp Definition. Retrieved 26 December 2014, http://www.w3.org/2008/webapps/

[27] Zusak, I, M. Ivankovic, and I. Budeselic, A Classification Framework for Web Browser Cross-Context Communication. Retrieved December 26, 2014 http://arxiv.org/abs/1108.4770