AD

AD-E404 176

Technical Report ARWSE-TR-15041

TACTICAL APPLICATIONS JAVASCRIPT DEVELOPMENT TOOLS RECOMMENDATIONS

Craig Klementowski
Tiffany Reid
Ross Arnold

January 2020

U.S. ARMY COMBAT CAPABILITIES DEVELOPMENT COMMAND ARMAMENTS CENTER

Weapons & Software Engineering Center

Picatinny Arsenal, New Jersey

Approved for public release; distribution is unlimited.

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

The citation in this report of the names of commercial firms or commercially available products or services does not constitute official endorsement by or approval of the U.S. Government.

Destroy by any means possible to prevent disclosure of contents or reconstruction of the document. Do not return to the originator.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-01-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden to Department of Defense, Washington Headquarters Services Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (*From - To*)
January 2020	Final Report	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Tactical Applications JavaScript Development Tools Recommendations	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHORS Craig Klementowski, Tiffany Reid, and Ross Arnold	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army CCDC AC, WSEC Fire Control Systems and Technology Directorate (FCDD-ACW-FM) Picatinny Arsenal, NJ 07806-5000	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
U.S.Army CCDC AC Knowledge & Process Management Office (FCDD-ACE-K) Picatinny Arsenal, NJ 07806-5000	11. SPONSOR/MONITOR'S REPORT NUMBER(S) Technical Report ARWSE-TR-15041

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The current design of Tactical Applications lends itself to a hybrid web application architecture. In such an architecture, the client and server may be hosted on two completely different systems or the client may play host to server components to support disconnected operations. Recent front-end development of hybrid web applications mainly involve a combination of HTML5, CSS3, and JavaScript. The commercial and open source software product landscape that aids in development of front-end web applications is quite vast. The TacApps development team needs and requires software development products that have a proven track record, Integrated Development Environment integration friendly, ease of use, and cost effective. The TacApps development team reviewed web development products that covered the following areas: static code analysis, code style enforcement, unit testing and continuous integration (CI) as it pertains to our environment for TacApps. Recommendations as a result of those reviews have been made in all four categories as follows: JSHint – static code analysis, JSCS (Java Script Code Style) – code style enforcement, pairing of Karma & Jasmine – unit testing, and Jenkins – CI. The containing document highlights all products reviewed and brings forth product recommendation with rationale.

15. SUBJECT TERMS

Tactical Applications Software development Software tools JavaScript Static code analysis Unit testing Coding style

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ross D. Arnold
U	U	U	SAR	19	19b. TELEPHONE NUMBER (Include area code) (973) 724-8618

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

CONTENTS

	Page
Introduction	1
Static Code Analysis	1
JSLint	1
Google Closure Compiler	1
JSHint	2
ESLint	2
JSXHint	2
Cascading Style Sheets (CSS) Lint	2
JavaScript Code Style Checker	2
JavaScript Code Styler (JSCS)	2
JS Beautifier	3
Unit Testing	3
Karma	3
Chutzpah	3
Jasmine	3
Mocha	3
QUnit	4
Continuous Integration Servers	4
Jenkins	4
Travis	4
Bamboo	4
Conclusions	4
Continuous Integration Recommendation	5
Unit Testing Recommendation	5
JavaScript Code Styler Checker Recommendation	5
Static Code Analysis Recommendation	5
Appendix – Backup Data	7
Bibliography	11
Distribution List	13

ACKNOWLEDGMENTS

The authors would like to thank Timothy Rybarski and Gregory Roehrich for their sponsorship as well as the Tactical Mission Command Product Management Office for funding the Weapons and Software Engineering Center to undertake this effort.

INTRODUCTION

At the time of the writing of this document, the Tactical Applications program has gone through critical design review. The TacApps development team faces time critical decisions regarding development and/or environment tools. The TacApps development team has had the opportunity to explore many useful software development tools as part of the design and technology investigations. Development toolsets encompassing unit testing, code analysis, code style enforcement, and continuous integration (CI) must be selected so that the development team involved in front-end development is working from the same product set to drive development synergy. This document will examine leading development tools that could be used for TacApps front-end development and processes surrounding such development. In order to easily collaborate with various stakeholders at disparate locations, the Defense Intelligence Information Enterprise (DI2E) platform is being used. The DI2E provides JIRA software, Confluence, Jenkins, Nexus, Fortify, Stash, Subversion, FishEye, and Storefront web applications to provide a central software development stack that can be used by multiple collaborating teams. The TacApps program is initially planning on using Stash, JIRA, FishEye and Jenkins tools. Therefore, other tools specifically for the use of source control management, issue management, and code review will not be evaluated.

STATIC CODE ANALYSIS

Static program analysis is the analysis of computer software that is performed without actually executing the program. In most cases, the analysis is performed on some version of the source code; in other cases, the analysis is performed on some form of the object code. The term static program analysis usually refers to the analysis performed by an automated tool whereas analysis performed by humans is referred to as program understanding, program comprehension, or perhaps code review. The word or term “lint” is often synonymous with static analysis by many as Lint was an actual early tool used for static analysis of programs written in the C language.

Static code analysis is a useful process for finding programming errors that may not be evident on the surface. Performing static code analysis helps to enforce coding conventions that are used in the development process. Execution of static analysis tools should be integrated as part of a normal build process.

JSLint

The JSLint was one of the first JavaScript code analysis tools made available. It was created by Douglas Crockford, a major JavaScript contributor best known for creating JavaScript Object Notation format. It is not configurable, customizable, or extensible. JSLint is not under active development and is infrequently updated by single contributor. A common criticism of JSLint is that it has strong opinions about certain coding standards that may create excess warnings.

Google Closure Compiler

Google Closure Compiler is a tool that simply makes JavaScript download and run faster by compiling JavaScript to efficient JavaScript performing the following activities in that compilation process: eliminating dead code, rewriting, and minimizing the remaining code. During this process, warnings and errors against the code are produced. Google Closer Compiler is updated frequently with a handful of major contributors. Google Closer Compiler is implemented in Java and is distributed as a single jar.

JSHint

The JSHint is a fork of JSLint allowing for customization options, which according to many developers was of extreme importance due to JSLint limitations. The JSHint appears to be the most popular static code analysis tool for JavaScript at this time. The JSHint is updated frequently with a handful of major contributors.

ESLint

The ESLint is quite similar to JSHint. One of its most valued features is that it allows for the addition of new rules at runtime with the ability to toggle those rules for execution. The main author admits that it runs two to three times slower than JSHint.

JSXHint

The JSX is an Extensible Markup Language (XML) like syntax extension to ECMAScript. It is used by preprocessors to transform this syntax into ECMAScript. TacApps CDR Demonstration used React, a JavaScript library for user interfaces and React JSX, which is a transpiler that transforms JSX into React JavaScript elements. The JSXHint was also used as part of demonstration development process. JSXHint is a wrapper for JSHint to allow “linting” of files that contain JSX.

Cascading Style Sheets (CSS) Lint

The CSS Lint offers “linting” for Cascading Style Sheets (CSS). CSS Lint is stable and is not under active development, but occasional changes have been made. There have been well-versed CSS developers that view some CSS Lint rules as “objectionable” but for our current TacApps team, there is high value in using this tool.

JAVASCRIPT CODE STYLE CHECKER

Enforcing a formalized coding style helps to maintain a consistent look and feel of the code. New developers or seasoned developers will be able to be productive on various pieces of code that were perhaps initially developed by someone else and will not be distracted by personalized code stylings. Some static code analysis tools offer limited code style checking but are far from complete. However, there are a few tools that can help maintain code common look and feel. The tools explored were JavaScript Code Style checker (JSCS) and JS Beautifier.

JavaScript Code Styler (JSCS)

The JSCS is clearly the leader in this area. JSCS programmatically enforces style guides (auto-formatting). It is configurable with over 90 validation rules which include presets from popular style guides, for example, jQuery, Google and Node. Integrated development environment (IDE) and other tooling support is constantly growing, which makes the integration of JSCS in most development environments seamless. Some major users of JSCS include: Adobe, Bootstrap, AngularJS, Ember.js, Grunt, and jQuery.

JS Beautifier

JS Beautifier cleans up “messy,” “minified,” and “obfuscated” JavaScript code. In addition, the tool can clean up CSS and Hypertext Markup Language (HTML) code. This could be used to address new code as you develop or as a precursor to JSCS. It is configurable by passing command line arguments for newline preservation, indentation size, indentation character, padding, and many more.

UNIT TESTING

Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine whether they are fit for external use (outside of said unit).

Unit tests for JavaScript are executed by a test runner. A test runner will execute tests coded in one of the various testing frameworks. Karma and Chutzpah are examples of test runners while Jasmine, Mocha, and QUnit are examples of unit testing frameworks.

Karma

Karma is the preeminent JavaScript test runner. Karma is a simple tool that allows for the execution of JavaScript code in multiple real browsers and real devices (phones and tablets). Karma easily integrates with common IDEs (i.e. WebStorm) as well as CI environments such as Jenkins, Travis, and Semaphore. It is a testing framework agnostic with built-in workflow control that integrates into an IDE.

Chutzpah

Chutzpah is a .Net-based command line test runner that can also be easily integrated into Visual Studio. It uses PhantomJS, which is a headless browser to execute tests. Not only is it a test runner, but it also reports on code coverage after tests have been executed.

Jasmine

Jasmine is a Behavior Driven Development testing framework for JavaScript. It does not rely on browsers, Document Object Model, or any JavaScript framework. It is suited for websites, Node.js projects, or anywhere that JavaScript can run. The syntax is extremely clean so that tests can easily be written.

Mocha

Mocha is a feature-rich JavaScript test framework running on Node.js and the browser, which facilitates the ease of asynchronous testing. Mocha provides test coverage reporting, file watcher support, highlights slow tests, test specific timeouts, and reports test durations to name a few features. Tests can be ran serially while mapping uncaught exceptions to correct test cases.

QUnit

QUnit is a powerful, easy-to-use JavaScript unit testing framework. It is used by the jQuery project to test its code and plugins but is capable of testing any generic JavaScript code as well as testing JavaScript code on the server-side. The assertion methods of QUnit actually follow the CommonJS unit testing specification, which was originally influenced by Q-Unit.

CONTINUOUS INTEGRATION SERVERS

The CI describes a set of software engineering practices that speed up the delivery of software by decreasing integration times. Software that facilitates this practice is called CI software. The top open source leaders in this space are Jenkins and Travis. Many GitHub-hosted products use Travis since the online-hosted service is free for open source projects. Jenkins appears to be the most widely used CI server and is easily hosted locally to satisfy enterprise solutions. There are also numerous commercial solutions that were not part of this evaluation.

Jenkins

Jenkins CI is the leading open source CI server. Built with Java, it provides over 1,000 plugins to support building and testing virtually any project. DI2E is using Jenkins as its CI server. Jenkins, previously known as Hudson, was forked and renamed after the Oracle acquisition. Jenkins provides the following tool integration with additional tools not listed: Eclipse, NetBeans, IntelliJ, Google Chrome, Firefox, Bash, and Ant.

Travis

Travis CI is hosted online by a company based in Germany. A free downloadable package for local hosting of Travis is not offered. Travis offers commercial installation for a fee. Theoretically, one could download code from GitHub and build the Travis CI server: however, that would be a significant undertaking and may not be worth it when Jenkins is a clear and viable solution.

Bamboo

Bamboo is a commercial solution from Atlasian, the provider of many DI2E solutions. Bamboo purported benefits include “best” JIRA integration and deployment support. IDE support in WebStorm is provided by IntelliJ Connector plugin. Perhaps DI2E may include Bamboo at a later date.

CONCLUSIONS

There are many mature tools available to aid in the development of TacApps. This document offered a brief glimpse into some tools that were used or investigated during the pre and post-CDR explorations. The tool recommendations largely follow current industry standards and best practices. Many tools can become interchangeable as the TacApps development team’s needs evolve while nuances of each tool are discovered.

Continuous Integration Recommendation

Jenkins

Using Travis would not seem to be a good fit for a couple of different reasons. TacApps is not an open source project. Hosting Travis locally or online would add additional costs to the TacApps project budget. Jenkins appears to be the best choice given that TacApps already uses Defense Intelligence Information Enterprise (DI2E) and the hosted version of Jenkins is readily available on DI2E. If decided upon, Jenkins could also be set up locally.

Unit Testing Recommendation

Karma and Jasmine

The pairing of Karma and Jasmine is the defacto standard for JavaScript unit testing; it is the recommendation that the TacApps team use Karma and Jasmine for its collective JavaScript unit testing.

JavaScript Code Styler Checker Recommendation

JavaScript Code Style

The JSCS appears to be the obvious choice for usage of code style enforcement by TacApps developers. Next logical step would be for the TacApps team to collectively decide on an agreeable preset as a starting point.

Static Code Analysis Recommendation

JSHint

Most of the tools have plugins for the major build platforms such as: Grunt, Gulp, Gradle, Ant, and Maven. Given the apparent ubiquity of JSHint, it is recommended to start there for JavaScript static code analysis. There is very little cost in adding some of the previously mentioned tools to the build. A further exploration of integration of the static analysis tools to the build process might be made to further extend our knowledge.

APPENDIX
BACKUP DATA

Node.js

Node.js emerged as an enabling technology that has been utilized by web developers as a base tool for web site development. Node.js is an event driven, non-blocking I/O model that is built on top of Chrome's JavaScript runtime. It is able to serve up web contents easily on demand and provides a platform for JavaScript based tools to run within. Many open source projects utilize Node.js packaged with the following tools: JavaScript task runners such as Grunt or Gulp, Node Package Manager (NPM, Node.js library management system) and Bower (another library management system) to quickly assemble disparate assets together to build their web applications. Most of the tools analysed in this document were utilized via Node.js along with Grunt task runner. It should be noted that many of the tools discussed can also be exercised in the Integrated Development Environment (IDE), WebStorm which happens to be the TacApps team IDE of choice.

BIBLIOGRAPHY

1. "douglascrockford / JSLint," The JavaScript Code Quality Tool, GitHub, Web, <https://github.com/douglascrockford/JSLint>, 01 May 2015.

2. "google / closure-compiler," A JavaScript checker and optimizer, GitHub, Web, <https://github.com/google/closure-compiler>.

3. "jshint / jshint," JSHint is a tool that helps to detect errors and potential problems in your JavaScript code, GitHub, Web, <https://github.com/jshint/jshint>.

4. "eslint / eslint," A fully pluggable tool for identifying and reporting on patterns in JavaScript, GitHub, Web, <https://github.com/eslint/eslint>.

5. "STRML / JSXHint," Wrapper around JSHint for linting JSX files, 100% compatible with existing tools using JSHint, GitHub, Web, <https://github.com/STRML/JSXHint>.

6. Hartikainen, J. A Comparison of JavaScript LintingTools, SitePoint, Web, <http://www.sitepoint.com/comparison-javascript-linting-tools> 05 March 2015.

7. "CSSLint / csslint," Automated linting of Cascading Stylesheets, GitHub, Web, <https://github.com/CSSLint/csslint>.

8. Codacy Blog, Review of CSS LintingTools, Codacy Blog, Web, <http://blog.codacy.com/2014/06/25/review-of-css-linting-tools>, 25 June 2014.

9. Tutorial and Overview. JavaScript Code Style. JSCS. Web. 2015. <http://jscs.info/overview.html>.

10. "jscs-dev / node-jscs," JavaScript Code Style checker, GitHub, Web, <https://github.com/jscs-dev/node-jscs>.

11. "beautify-web / js-beautify," Beautifier for javascript, GitHub, Web. <https://github.com/beautify-web/js-beautify>.

12. "karma-runner / karma," Spectacular Test Runner for JavaScript, GitHub, Web, <https://github.com/karma-runner/karma>.

13. "mmanela / chutzpah," Chutzpah is an open source JavaScript test runner which enables you to run unit tests using QUnit, Jasmine, Mocha, CoffeeScript and TypeScript. GitHub, Web, <https://github.com/mmanela/chutzpah>.

14. Groat, K., "Which JavaScript Test Library Should You Use?" QUnit vs Jasmine vs Mocha, Tech Talk DC., Web, <http://www.techtalkdc.com/which-javascript-test-library-should-you-use-qunit-vs-jasmine-vs-mocha>,04 May 2014.

15. Kobletz, A., Shelajev, O., and White, O. "DevProd Report Revisited," Continuous Integration Servers in 2013, Rebellabs, Web, <http://zeroturnaround.com/rebellabs/devprod-report-revisited-continuous-integration-servers-in-2013>,12 February 2013.

16. Docs. Hudson Extensible continuous integration server. Web. <http://hudson-ci.org>, 2015.

BIBLIOGRAPHY
(continued)

17. Use Jenkins. Jenkins, An extensible open source continuous integration server. Web, <http://jenkins-ci.org>, 2015.

18. Documentation, Test and Deploy with Confidence, Web, <https://travis-ci.org>, 2015.

19. 10 Reasons You'll Love Bamboo. Bamboo. Atlassian, Web<https://www.atlassian.com/software/bamboo/got-jenkins>, 2015.

20. http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#JavaScript

21. http://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software

22. http://stackoverflow.com/questions/24391462/what-are-the-differences-between-mocha-chai-karma-jasmine-should-js-etc-te

DISTRIBUTION LIST

U.S. Army CCDC AC
ATTN: FCDD-ACE-K
FCDD-ACW-HH, C. Klementowski
T. Reid
FCDD-ACW-FM, R. Arnold
Picatinny Arsenal, NJ 07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste 0944
Fort Belvoir, VA 22060-6218

GIDEP Operations Center
P.O. Box 8000
Corona, CA 91718-8000
gidep@gidep.org

REVIEW AND APPROVAL OF ARDEC REPORTS

THIS IS A:

☒ **TECHNICAL REPORT** ☐ **SPECIAL REPORT**

☐ **MEMORANDUM REPORT** ☐ **ARMAMENT GRADUATE SCHOOL REPORT**

FUNDING SOURCE PM funded EMD
[e.g., TEX3; 6.1 (ILIR, FTAS); 6.2; 6.3; PM funded EMD; PM funded Production/ESIP; Other (please identify)]

TacApps JavaScipt Development Tools Recommendations
Title

Tactical Applications
Project

Ross Arnold
Author/Project Engineer

Report number/Date received (to be completed by LCSD)

x8618 — Extension
59 — Building

RDAR-WSF-M
Author's Office Symbol

PART 1. **Must be signed before the report can be edited.**

a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.

b. Use Distribution Statement A_x_, B____, C____, D____, E____, or F____ for the reason checked on the continuation of this form. Reason: _____

1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.

2. If Statement B, C, D, E, or F is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.

c. The distribution list for this report has been reviewed for accuracy and completeness.

Tri Lu
Division Chief 6/28/2017 (Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

Tri Lu
Division Chief 6/28/2017 (Date)

RDAR-CIS (Date)

LCSD 49 (1 Sept 16)
Supersedes SMCAR Form 49, 20 Dec 06