Capitalizing on App Development Tools and Technologies

By Kenneth J. Luterbach, East Carolina University and
Kenneth R. Hubbell, Ingersoll Rand University

Abstract

Instructional developers and others creating apps must choose from a wide variety of app development tools and technologies. Some app development tools have incorporated visual programming features, which enable some drag and drop coding and contextual programming. While those features help novices begin programming with greater ease, questions arise about the overall utility of visual tools for app development. Analyses and comparisons of app development tools and technologies would make their advantages and disadvantages apparent, enabling instructional technologists to make informed decisions about tool selection. Toward that end, this work generated a new fram ework for comparing app development tools. The criteria that comprise the framework were then used to compare multiple authoring tools and technologies for creating apps. Three app development tools, namely Corona, LiveCode and MoSync, emerged as particularly noteworthy for their utility and flexibility, and because they are free of monetary cost or have a free version.

Keywords: app development tools; computational thinking; instructional apps

Technological changes to development tools continually present challenges and opportunities for instructional technologists and others. Presently, the availability of a wide variety of app development tools and technologies invites consideration along multiple paths, some of which are evident in the following questions. Does everyone have essentially the same conception of an app? What benefits accrue to those who learn to develop apps? What costs do those learners incur? Should everyone learn to develop apps? What similarities and differences exist among app development tools? Is there a particularly good app development tool? Is there an authoring tool that enables developers to create apps for both desktop and mobile platforms? Is there a particularly good app development tool for creating instruction? Answers to those questions will provide insights into how educational technologists may capitalize on app development tools.

Some scholars and commentators assert that everyone should engage in computer programming in order to develop *computational thinking* ability (Grover & Pea, 2013; Resnick, 2012; Prensky, 2008; Wing, 2006). Computational Thinking (CT) is regarded as essential for literacy, which puts CT on the same level of importance as reading, writing and arithmetic. Essentially, the claim is that a literate citizen must have developed sufficient CT capacity (including, for instance, abstraction, information representation and algorithm design) to solve a wide variety of problems that arise in many disciplines (Abelson, Turbak, Morelli, Martin, & Wolber, 2012; Grover & Pea, 2013; Wing, 2006). Importantly, designing

and implementing a variety of computer programs develops CT. Work to enhance the CT capacity of the general populace continues through initiatives such as the federally funded dissemination of instructional materials for *App Inventor* to promote computer programming as an activity for everyone (Abelson, Turbak, Morelli, Martin, & Wolber, 2012). *App Inventor* is a visual programming tool in which the developer drags, drops and connects visual blocks to create a computer program. In a related initiative, faculty and students in the Department of Computer Science at Wellesley College, boosted by messages from President Obama and myriad celebrities, seek to engage ten million people in one hour of coding (Wellesley College, 2013).

There is a growing base of literature that documents early efforts to teach development of web apps (Hsu & Ching, 2013; Martin, Pastore, & Snider, 2012) and diffuse augmented reality development software (Holden, 2014; Martin, Dikkers, Squire, & Gagnon, 2014), but literature on particular app development tools and processes is still lacking. Missing are analyses and descriptions of general-purpose development tools and a framework for comparing tools. This article seeks to address those gaps. First, this article ensures a shared definition or conception of apps. Second, this article discusses the motivation for learning to develop apps. Third, this article provides a framework for comparing app development tools and technologies and compares twelve such tools and technologies.

Conceptions of Apps

For breadth of understanding, it is beneficial to bring multiple perspectives to the conception of *apps*, which is actually a one-syllable abbreviation for *applications*, as in computer applications or computer programs. One valid conception of an app is that it is a computer program downloaded from an app store or service, which runs on a mobile device, typically a handheld device with screen size approximately 3" by 5" (phone size) or about 6" by 8" (tablet size). Less common today are computer apps for wearable devices, such as wristwatches and eyeglasses. One may also conceive of a program running on a desktop or laptop computer as an app. An app is a computer program of direct benefit to users. An app may help a user complete a task or entertain a user. Apps are often interactive, which require user input, but in the case of an app to play music in a radio format, the app requires no interaction beyond opening it with the mere touch of a finger. Though apps run in accordance with operating system software, system software is not regarded as an app. In addition to those views of apps, developers bring additional perspectives that provide a more complete conception of apps. Some app developers may even wonder at times whether to conceive of a website as an app.

For a robust view of apps, it is helpful to distinguish between *web apps* and *native apps*. In the early 1990s, when World Wide Web servers and browsers (clients) first appeared, websites delivered content. Users clicked on hyperlinks to read text; look at images; listen to audio; and view videos. Soon developers sought to enable user interaction through web pages. Through a Common Gateway Interface (CGI), web page and website developers began to present forms that users completed. Such functionality lead, for instance, to online shopping and banking, to blogging, to crowdsourcing in wikis, and to virtual frog dissection for learners. Over time, the technologies for user interaction on the web became more sophisticated and some standardization occurred. In this light, it is possible to write an app with web technologies, namely HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. In addition, through standardized libraries of JavaScript code (e.g., jQuery, jQueryUI and jQueryMobile), developers of web apps can include functionality created by others. For example, a JavaScript developer can use a jQuery function to animate an object or a jQueryUI object to implement a progress bar. Developers leverage the functionality available in code libraries through an Application Programming Interface (API), which defines the syntax required to incorporate a feature in the code library.

A developer of a web app may wish to use functionality in a code library to develop an interactive instructional story or game, for instance. Such a developer may use Undum (undum.com), which is a framework for interactive fiction using web browser (client-side) technology. Similarly, Vorple (Vorple-if.com) is a library of JavaScript routines for implementing interactive fiction. Developer tools for creating web apps include text editors (e.g., Notepad, TextEdit, Sublime Text) and software dedicated to web page development (e.g., Adobe Dreamweaver, Microsoft FrontPage).

The use of web technologies for deploying apps is effective in many cases and offers considerable platform independence. That is, a well-tested and refined web app will function in a web browser running under Mac OS,

Microsoft Windows, Linux, iOS, Android and other operating systems, although web browsers (e.g., Chrome, Firefox, Internet Explorer, Opera and Safari) are not equally effective. App development for universal deployment through web browsers requires testing across multiple devices (including different versions of the devices), distinct operating systems (including different versions of operating systems) and four or five popular web browsers (including different versions of web browsers). Data gathering and usage metrics available through services like Google Analytics are another benefit to developers of web apps because the data enable tracking of all devices through a single record set. In some cases, one disadvantage of implementing instruction using web technologies is that direct access to operating system functionality is lost. Consequently, accessing data from an accelerometer or gyroscope, for instance, requires additional custom development or may not be available at all. Some app development tools provide access to device functionality through a hybrid system, which enables developers to deploy apps on the web and use some features native to the operating system of the device. One variation on this theme is an app, much like a web browser, though which developers deploy their apps. Creating a web app, an app for deployment through a web browser, is one viable option for developers.

Alternatively, app developers may use APIs dedicated to a particular operating system in order to create native apps. In this way, for instance, any Apple iPhone or iPad app developer can make use of precisely the same code for performing tasks, such as playing video through the iOS video controller or opening a file through the File Open dialog, as a programmer at Apple. Consequently, use of native APIs makes it possible for an app developer to provide the *look and feel* commonly experienced by users of a particular platform (i.e., device and operating system). Apps for Android devices draw on code in Google's Android API. APIs also exist for developing apps that run on the Kindle and Blackberry platforms, for instance. Further, some APIs enable development of apps for wristwatches and eyeglasses, among other devices. While creating a native app, a developer may also leverage code in special purpose APIs, such as Google's API for displaying a map of a particular address, or any of the numerous commerce APIs for processing a payment given credit card information. A common API for apps that makes intensive use of graphics is Open GL (GL for Graphics Library) and Open GL ES (ES for Embedded Systems). The TinCan Experience API (see SCORM.com for more details) enables organizations and institutions to track individual student performance. Of the more than 1000 APIs (more than 200 in the education category according to one API tacking service, http://www.programmableweb.com/apis/directory/1?apicat=Education), certain ones are available only in particular development environments and only for specific target platforms (e.g., web browsers, iOS devices and Android devices). As evident in the section on comparing app development tools, multiple authoring tools are available to developers of native apps. Some authoring tools restrict the development of native apps to a particular platform (e.g., XCode for iOS, App Inventor for Android) while other tools enable deployment across multiple platforms (e.g., Corona, LiveCode and MoSync).

Motivation for Learning to Develop Apps

The proliferation of apps on mobile devices has moved and continues to move people throughout the world further into the information era, in which users have also become creators. Such recognition enables one to gain a sense of the extent to which apps have impacted many people worldwide. Can an instructional app have such widespread impact? Who can say what one instructional app alone might accomplish? However, there are nearly 3000 apps classified in the Education category in Apple's App Store and thousands more educational apps in Google Play and thousands more in app stores at companies such as Microsoft, Intel and Amazon, as well as an Educational App Store website (www.educationalappstore.com) that provide additional commentary and sorting of educational apps. Even though some educational apps across the app stores are duplicates, with over 190 million mobile app downloads per day (ABI Research, 2014), the app distribution infrastructure is an effective app deployment method. Learners, parents, teachers, app developers and others - particularly instructional technologists – may benefit from such an efficient method for offering instruction. While recognizing that an instructional app may reach a large number of learners, it is also helpful to recognize alternative uses of app development tools.

Some app development tools enable the development of multimedia presentations without any knowledge or skill in computer programming. These app development tools feature visual user interfaces, which enable one to create slides or "screens" or "cards" to display

text and images, as well as to play audio and video. These features provide an alternative environment to PowerPoint or Prezi for K-12 teachers or college professors who may assign students the task of assimilating or synthesizing content for presentation by having their students create an app. LiveCode is an example of such an app development environment, as is XCode, at least to some extent. There are some limitations in current versions of LiveCode and XCode; the inclusion of formatted text is not as easy as it is in PowerPoint, for instance. However, this loss of functionality is offset by not having to engage in computer programming to display text in a window.

Since app development tools enable computer programming, learners and others may use them as outlets for creative expression and problem solving (Luterbach, 2013a; Resnick, 2012). For some learners, the goal may not be the development of an app for widespread distribution, but learning how to solve a problem, such as one in machine learning, which can be done in as few as ten lines of code, a coin problem (Demaine, Demaine, & Verrrill, 2002; Luterbach, 2013b), or to control robots like Lego Next. Further, learning app development skills for desktop and mobile devices will enable positive transfer of learning to other development environments in which a developer may encounter a software development kit for wristwatches or eyeglasses. This transference also supports peer-to-peer learning opportunities for mentoring and coaching.

Comparing App Development Tools

The diversity of app development tools affords instructional developers a variety of options for creating instructional apps. This diversity, however, creates challenges for determining which development tool to learn and use. This section compares authoring tools designed solely for developing apps, which excludes consideration of Adobe Captivate 7, which is a software tool that added app development to an existing program for developing computer-based instruction. Even though it is possible to create apps with Adobe Captivate 7 (Adobe, 2013), which is well suited to the development of some computer-based instruction, at this time use of Captivate 7 for app development is limited because it was not designed to function as an all-purpose app development environment. In addition to requiring that the tool be designed solely for developing apps, this analysis sought to compare app development tools that are free of cost; target a variety of platforms, including Android, iOS, Kindle, Blackberry, Windows Phone, Mac OS, Linux, and Windows; and enable different forms of app development, whether through visually connecting blocks, dragging icons to a stage, or coding in a text editor.

Framework for Comparing App Development Tools

This new framework for comparing app development tools accounts for research on the diffusion of innovations. In particular, Rogers (2003) identified five factors as having significant influence on decision-making about whether to accept an innovation. Those factors are relative advantage, complexity, observability, trialability and compatibility to values. This analysis considers criteria that enhance or inhibit those diffusion factors. Specifically, the comparative framework includes the following criteria: (1) monetary cost, (2) ease of installation, (3) type of programming interface, (4) simplicity of the computer programming environment or Software Development Kit (SDK), (5) type of programming; (6) target platforms for deploying apps, (7) stability, (8) utility and adaptability for developing computer-based instruction, (9) instructional support for learning, and (10) sustainability/maintenance.

Regarding the type of programming interface, some authoring tools include a Visual Programming Interface (VPI). These interfaces enable the developer to drag and drop objects on to a window and set properties of the objects through dialogs. Further, in a VPI, developers use minimal to no scripting in order to connect objects. This analysis regards VPIs as easier to use than tools without that type of interface. In addition, instructional designers and others without programming experience consider authoring tools that permit contextual programming easier to learn and use than tools that require knowledge about placement of code in a source code file. For example, in contextual programming, the developer includes the code necessary to respond to a button click in a code window for the button. Another criterion this analysis uses to assess the complexity of the computer programming environment is the syntax of the language. This analysis regards use of code similar to the English language as simpler than syntax that focuses on symbols that do not form words in English.

With respect to utility and adaptability of the authoring tool for developing instructional

apps, this analysis considered whether the tool enables responsive (resizable) design for auto-fitting the interface objects to different screen sizes of devices running the app. Further, analysis of utility and adaptability considered the variety of user-interface elements (e.g., buttons, menus, scroll bars, windows, tables, lists, progress bars) available to developers, as well as the ease with which developers can implement and alter the objects. In addition, this criterion concerned the rapidity with which developers can cycle between testing and editing (a feature that some are calling "Live testing"). Stability concerns the frequency with which the development environment terminates unexpectedly (i.e., "crashes"). This analysis also considered the number and type of compatible app deployment devices. Of particular importance to novice developers, this comparative work assessed the availability and quality of instructional materials and forums in support of learning the tool. Lastly, keeping in mind the history of authoring and development tools in the learning industry (especially recognizing the demise of Authorware, Director and MoGlue to name a few), this analysis considered the sustainability and maintenance of the tool.

Comparative Analysis

Table 1 compares app development tools across the criteria discussed above; the ratings for particular criteria range from low (1) to high (5).

The evolution of app development tools is dynamic. The rating for the version of LiveCode that immediately preceded Version 6.5 would have rated perhaps 1 out of 5 on the utility attribute because automatic resizing of objects for variable screen sizes was not available. In the case of MoGlue (not in Table 1), the development and deployment platform was released in 2010 and by the end of 2013 the company had gone out of business, leaving their users without a means for continued development and maintenance. Corona has a visual programming interface in Beta testing and plans to add Windows Phone as a target platform. When will those development goals be attained? Will use of Lua (http://docs.coronalabs.com/guide/start/introLua/index.html) increase, hold steady, or decrease? What new features will LiveCode 7 include? Will a new JavaScript API be created to support or enhance instructional apps? While the future is uncertain, it is possible to address the current state of app development tools.

Contextual programming does have some limitations and developers must keep these in mind when creating large apps or apps built using portions of code from other apps. When developing a single standalone app, code reuse may not be very important; however, for organizations developing multiple learning apps, code reuse is critical to efficient production. Whereas developers of web apps routinely link to external JavaScript files, app developers using a contextual programming interface, unfamiliar with how code in a computer programming language is structured, may not understand how to link to external files in order to include features available through APIs. Developers can add critical functionality to their apps with minimal additional coding by extending their programs through APIs. Additionally, developers of web apps, unlike other app developers, can change an external CSS file to rapidly modify the look and feel of the entire interface by altering colors, fonts and other visual characteristics. Developers using JavaScript and HTML also benefit from over two decades of revisions to code libraries, tutorials and samples from which to draw inspiration and support.

Selection of an app development tool is often a difficult task based on available resources, budget, developer skills and, most importantly, the desired learning solution. The selection process is cyclical, going back and forth between desired functionality and assessment of the development tool's features. In certain cases, the development team may drive the selection process because, over time, they may have amassed a substantial code library that they use to create apps.

Another key factor affecting selection of the development environment is the target device. Some of the tools listed in Table 1 are limited to a few specific platforms. Other tools create apps that can be deployed everywhere. All of these tools run on Windows and Mac OS. However, to publish for iOS devices, developers must have a Mac available for final preparation. Further, iOS apps must also go through an approval process before they can be released through the Apple App Store. In contrast, Android apps can be distributed through the Android Market or through the web or local area networks.

Given those comparisons, instructors and instructional developers may select different development tools for different purposes. For example, MoSync Reload may be used for development and subsequent deployment of apps across a wide variety of devices while, depending on prior knowledge of the learners, LiveCode may be used for teaching how to develop apps.

Table 1. Comparison of App Development Tools

	Cost	Easy to Install	VPI	Ease of SDK	Type of Programming	Targets	Stability	Utility/ Adaptability	Support	Sustainability / Maintenance
App Inventor	Free	No install req.	Yes	2	Visual blocks	Android	3	1	3	4
LiveCode	Free	5	Yes	3	English commands with terse phrasing	Native Mac OS Windows Linux iOS Android	3	5 (requires installing APIs for iOS and Android)	3	4
Corona	Free	5	No	3	Lua	iOS Android Kindle	3	5	4	4
Phone Gap	Free	2	No – reqs HTML editor	2	HTML, CSS, Javascript	Native and Hybrid (Amazon Fire OS Android BlackBerry 10 iOS Windows Phone Windows 8 Tizen)	3	5 (requires installing APIs)	$249.99 to $19,999 based on level of service	4
ARIS	Free	n/a	Yes	5	n/a	All	3	1	2	1
MoSync	Free	5	No – reqs HTML editor	4	HTML, HTML5, CSS, Javascript	Native and Hybrid (Android iOS Windows 8 IE 10+ Chrome Safari Firefox Opera)	3	5 (requires installing APIs)	2	4
Vorple (Library can be used with MoSync)	Free	3	No – reqs HTML editor	3	HTML5, CSS, Javascript	Web-app (Android iOS Windows 8 IE 10+ Chrome Safari Firefox Opera)	3	5 (requires additional APIs and custom javascript libraries)	1	3
Wordpress	Free	3	No – reqs HTML editor	3	HTML or HTML5, CSS, Javascript	Web-app (Android iOS Windows 8 IE 10+ Chrome Safari Firefox Opera)	4	5 (requires installing APIs)	5	5

Continued on next page.

	Cost	Easy to Install	VPI	Ease of SDK	Type of Programming	Targets	Stability	Utility/ Adaptability	Support	Sustainability / Maintenance
Unity3D	Free to try + $1500 + $1500 per target	5	5	4	C# or Javascript	Native (Android iOS Windows 8 Blackberry 10 IE 9+ Safari Firefox Linux Mac Windows)	4	5 (requires installing APIs)	5	3
HTML5	Free	n/a	No – reqs HTML editor	3	HTML5, CSS, Javascript	Web-apps (browser must support HTML5)	4	5 (requires installing APIs)	3	3
Game Closure	Free	3	No – reqs HTML editor	4	HTML, HTML5, CSS, Javascript	Native (Android and iOS)	3	5 (requires installing APIs)	3	3
Rho Mobile	Free for open source apps	2	Built-In IDE with simulator	3	Javascript, HTML, CSS and Ruby (optional)	Native (Android iOS Windows 8)	3	5 (requires installing APIs)	$$	3
XCode	Free	5	Yes	2	Objective C	iOS	3	3	4	4

Summary and Conclusions

One should consider multiple factors before choosing an app development platform. First, the intended use of the app should drive the selection process. Second, cost is an easy way to eliminate some of the options quickly. Third, the specific skills and ability level of the developers will reduce the list. Fourth, the target device will further refine the list of options. At that point, the list is typically short and one can make the final selection based on the factors relevant to the app development project.

By applying the new framework for comparing app development tools and technologies, three development tools available at no cost emerged as noteworthy because they meet the needs of many instructional developers and students of instructional development. Those app development tools are Corona, LiveCode and MoSync.

Of the three particularly noteworthy development tools, only Corona offers prepackaged device simulation. In contrast, LiveCode and MoSync require the developer to spend time installing device simulators, which can be time consuming and, in some cases, frustrating. Corona is also unique among the noteworthy tools in offering two output targets, a text window and a device simulator. Corona developers can build apps for distribution on iOS and Android devices, as well as devices running Android derivatives, such as Kindle and Nook. One advantage to novices of both a textual output window and a device simulator is that simple 5 – 10 line programs can be written to learn computer programming fundamentals. The "Corona in 5 Minutes" tutorial (http://docs.coronalabs.com/guide/start/helloWorld/index.html) introduces learners to both output options. Even though the purportedly 5-minute tutorial may take approximately 50 or more minutes to complete, the tutorial enables first-time programmers to write code in order to create and run three apps, two of which are simulated on the device of the learner's choosing; the other, and first app, uses one line of code to display "Hello World" in the textual output window. Experienced programmers may appreciate the two output targets for enhancing debugging by inspecting values of variables in the text output window while testing the app in a device simulator. Corona developers create apps using a text editor to write statements in the

Lua scripting language. Developers can readily move between text editing and testing, which is important for instructional development.

LiveCode may appeal to visual developers in particular. The drag and drop interface for including text, images, sound and video affords the developer the opportunity to create an app without any computer programming. When a developer does want to include code to make a more sophisticated app, the coding is contextual. That is, the text-editing window in which the developer enters code is associated with the object. In LiveCode, developers use statements similar to the English language and functional apps may have few statements. For example, a button can respond to a mouse click or finger tap and take the user to the next screen (called a card in LiveCode) by entering *go to the next card* in the code window of the button. To enable a user to drag and drop a button, or any other visual object, the developer could put *grab me* in the code window of the object. Surrounding *grab me* by *on mouseDown* and *end mouseDown* would complete the coding and enable users to drag and drop objects. To animate an object, the developer would use a *move* command. LiveCode enables developers to readily switch between editing and testing. Through visual and contextual programming, LiveCode enables novices with no computer programming experience to begin developing apps quickly.

MoSync may be especially appealing to instructional technologists and others who have learned HTML, CSS and JavaScript. With knowledge of those technologies, a developer can, at once, create an app to run in the MoSync client app on mobile devices and in web browsers on desktop and laptop computers. Deploying apps to multiple mobile devices in MoSync is simple and fast, which makes for quick transitions between editing and testing. Given that JavaScript is used to create apps, developers can leverage the many stable extension libraries to enhance functionality. For example, instructional developers may use Undum and Vorple to engage learners through interactive fiction.

Learning any one of those three authoring tools will enable instructional developers to create multiple learning solutions. The availability of excellent sample programs and thriving developer communities for Corona, LiveCode and MoSync reduces the path to competency. Though weak documentation is a shortcoming for both MoSync and LiveCode, when necessary, one may utilize the excellent MoSync and LiveCode forum communities or readily find tutorials and additional information through other online sources. MoSync uses freely available technologies, namely HTML, CSS and JavaScript. LiveCode and Corona are different. The Community version of LiveCode is free of cost and open source. Though free of cost, apps created with the Community version must also be open source (http://livecode.com/how-it-works/open-source/). The commercial version of LiveCode costs $500 annually; LiveCode Pro is $999 annually. The Starter version of Corona is free of cost, but additional features are available for annual fees, per developer, of $192, $588, $948, and $2388 depending on features and level of support. Access to native API/libraries costs a Corona developer $948 annually. The Beta version of the visual programming interface for Corona is available at the $588 pricing level (http://coronalabs.com/pricing/).

The app development enterprise is broad and dynamic, as well as open to everyone who wants to try developing an app. One starting point is the "Hour of Code" tutorial (www.code.org), which over 35 million people have tried. We encourage instructional technologists and others to learn fundamentals of computer programming to foster computational thinking (Grover & Pea, 2013; Wing 2006). Additionally, app development enables one to gain personal perspectives on contemporary society, which is influenced by software development (Manovich, 2013). For novice developers, reusing and adapting ("hacking") the examples of masters is important. Developing instructional apps involves the synthesis of content knowledge, pedagogical knowledge and technical skills. With respect to instructional app development, one must acquire technical skills regarding particular computing devices, operating systems and at least one app development tool. In such light, instructional developers and the educational technology field would benefit from additional comparisons of app authoring tools, especially as applied to the development of instruction. Consideration of server side technologies, APIs and browser plug-ins of particular import to educators would also advance the field. Discussions and demonstrations of innovative instructional apps in Science, Technology, Engineering, and Mathematics (STEM fields), humanities, arts and other fields would advance those fields as well as educational technology.

There are no perfect app development tools suitable for every instructional development project or for everyone learning programing to enhance their computational thinking. However, the lack of one superior development environment should not deter one from learning to develop apps. Opportunities abound for instructional developers and others with app develop-

ment skills for any platform. It is important for instructional developers to keep up-to-date with tools, new platforms and changes to the target devices. Keeping current allows developers to take advantage of the latest features and, ultimately, to deliver learning solutions wherever needed.

Kenneth J. Luterbach *is an Assistant Profssor in the Department of Mathematics, Science, and Instructional Technology Education in the College of Education at East Carolina University, Greenville, NC. You may direct correspondence regarding this article to him via email at: luterbachk@ecu.edu.*

Kenneth R. Hubbell *is Senior Manager, Learning Technology at Ingersoll Rand University in Davidson, NC. You may email him at: khubbell@irco.com*

References

Abelson, H., Turbak, L. Morelli, R., Martin, F., & Wolber, D. (2012). NSF awards grant for App Inventor Teaching Project. Retrieved January 11, 2014, from http://appinventor.mit.edu/explore/news/nsf-awards-grant-app-inventor-teaching-project.html

ABI Research (2014). Android will account for 58% of smartphone app downloads in 2013, with iOS commanding a market share of 75% in tablet apps. Retrieved May 12, 2014, from https://www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down

Adobe (2013). Adobe captivate app packager. Retrieved January 11, 2014, from http://helpx.adobe.com/captivate/using/captivate-app-packager.html

Demaine, E. D., Demaine, M. L., & Verrill, H. A. (2002). Coin-moving puzzles. *More games of no chance*, *42*, 405-431.

Grover, S. & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, *42* (1), 38 – 43.

Holden, C. (2014). The local games lab ABQ: Homegrown augmented reality. *TechTrends, 58* (1), 42 – 48.

Hsu, Y. -C., & Ching, Y. -H. (2013). Mobile app design for teaching and learning: Educators' experiences in an online graduate course. *The InternationalReview of Research in Open and Distance Learning*, *14*(4), 117-139.

Luterbach, K. J. (2013a). Building software development capacity to advance the state of educational technology. *Educational Technology*, *53* (2), 21–27.

Luterbach, K. J. (2013b). Elegant Instruction. *Journal of Educational Technology Systems. 41* (2), 183-204.

Manovich, L. (2013). Software takes command. New York, NY: Bloomsbury Academic.

Martin, F., Pastore, R., & Snider, J. (2012). Developing mobile based instruction. *TechTrends, 56* (5), 46 – 51.

Martin, J., Dikkers, S., Squire, K., & Gagnon, D. (2014). Participatory scaling through augmented reality learning through local games. *TechTrends, 58* (1), 35 – 41.

Prensky, M. (2008). Programming is the new literacy. *Edutopia*. Retrieved January 11, 2014, from http://www.edutopia.org/programming

Resnick, M. (2012). Reviving Papert's dream. *Educational Technology, 52* (4), 42-46.

Rogers, E. (2003). *Diffusion of innovations (5th edition)*. New York, NY: Free Press.

Wellesley College (2013). The hour of code. Retrieved January 11, 2014, from https://www.youtube.com/watch?v=FC5FbmsH4fw and http://www.wellesley.edu/news/2013/12/node/40715