Beyond Open Source: The TouchDevelop Cloud-based Integrated Development Environment

Thomas Ball, Sebastian Burckhardt, Jonathan de Halleux, Michał Moskal, Nikolai Tillmann
Microsoft Research
One Microsoft Way, Redmond WA 98052, USA
{tball,sburckha,jhalleux,micmo,nikolait}@microsoft.com

***Abstract*—Software engineering tools and environments are migrating to the cloud, enabling more people to participate in programming from many more devices. To study this phenomenon in detail, we designed, implemented and deployed TouchDevelop (www.touchdevelop.com), a cloud-based integrated development environment (CIDE), which has been online for the past three years.**

TouchDevelop combines a cross-platform browser-based IDE for the creation of mobile+cloud apps, an online programmer/user community, and an app store. A central feature of TouchDevelop is to track all program edits, versions, runtime information, bugs, as well user comments, questions and feedback in a single cloud-based repository that is available publicly via Web APIs.

In this paper, we examine a key feature of TouchDevelop that should be relevant to others creating CIDEs, namely the seamless integration of replicated workspaces, simplified version control and app publishing. An analysis of the TouchDevelop repository shows that this combination of capabilities allows users to easily create new versions of apps from existing apps, make changes to other users' apps, and share their results from a variety of devices, including smartphones, tablets and traditional PCs.

I. Introduction

Application (app) stores for mobile platforms such as tablets and smartphones are very popular. Even though the majority of apps available in these stores are conceptually very simple, there can be a lot of friction in their development, deployment, and maintenance. The hopeful app creator must first master a complex set of technologies, including general purpose programming languages, integrated development environments (IDEs), and version control systems, to name a few. The next step is understanding the deployment and update models of the various app stores. Finally, many mobile apps require a cloud back-end to store user data and telemetry data not provided by the app store, which requires gaining mastery of cloud infrastructure. For many people, this puts mobile app creation out of reach.

Three years ago, we created a language and programming environment called TouchDevelop [1] (www.touchdevelop.com) to make it easy for non-experts to author mobile apps. TouchDevelop integrates a client IDE, web-based version control with a programmer community (as in GitHub [2]), and an application store in a seamless experience, specialized for a single programming language.

Over the past three years, the client IDE for TouchDevelop expanded from a smartphone app for Windows Phones into a Web app running in all modern HTML5 browsers on phones, tablets, and keyboard-equipped PCs (regardless of the operating system they may be running). The TouchDevelop cloud back-end provides replicated workspaces for authenticated users, a version control system with automated dependency tracking, an app store, as well as a crash-logging system with bucketization, and crowd-sourced coverage collection. All data published by the community (apps, comments, bugs, etc.) is available publicly via Web APIs.[1]

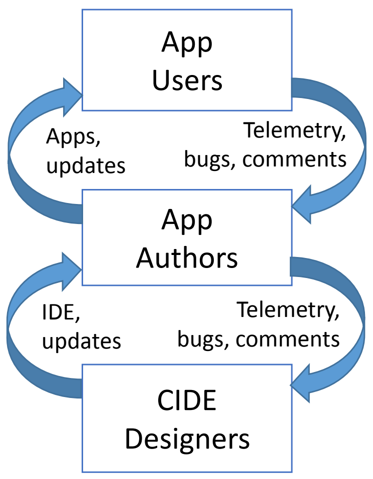


Fig. 1. Positive feedback cycles between CIDE designers, authors and users.

TouchDevelop is an instance of a *cloud-based integrated development environment* (CIDE), bringing together three audiences, as shown in Figure 1:

- **CIDE Designers** are the authors of this paper. The TouchDevelop IDE is delivered as a Web app. We use the cloud to deliver multiple versions of the IDE simultaneously; TouchDevelop users who have a high experience score are prompted to use the *beta* version, which includes new features not in the *current* version. The IDE has a high level of instrumentation and telemetry that sends data back to the cloud to help us understand the stability of a new release and the usage of features.

[1] See https://www.touchdevelop.com/help/cloudservices

Assertion failures and unhandled exceptions in the IDE automatically generate bug reports, allowing us to quickly see if a new feature is causing problems.
- **App Authors** use the TouchDevelop IDE to rapidly build apps and iterate with their users, as well as with the CIDE designers to get bugs fixed and/or questions answered. Furthermore, just as the TouchDevelop IDE contains instrumentation, the TouchDevelop compiler inserts instrumentation into apps to automatically collect and store profile and coverage information in the cloud. Runtime failures in a TouchDevelop app present users with a dialog whereby they can submit a stack trace to the app author via the cloud.
- **App Users** can provide feedback to authors in comments and bug reports, as mentioned above. But, more importantly, *app users are empowered to become app authors* as the editing functionality is only one tap away when running the app and re-publication of modified apps takes just a few seconds. TouchDevelop tracks the provenance of apps, so that the author of an app that is subsequently changed and re-published by others receives an increased ranking.

In this paper, we present the design of TouchDevelop's cloud back-end and how it simplifies the creation, maintenance and publication of apps. In particular, we believe that for beginning app authors, fully featured version control systems together with app store publication models are too complex. The TouchDevelop cloud seamlessly integrates the three services of (1) automated workspace replication, (2) version control with automated dependency tracking, and (3) an app store. Workspace replication provides authors with a private and consistent view of the apps that they are working on, no matter which device they connect to TouchDevelop from. Version control with dependency tracking means that authors need not worry about which set of files/resources form a consistent snapshot of an app - the system ensures that when an author publishes an app to the public version control, that all transitive dependencies are published as well. Finally, the app store makes it easy for authors to update their apps and deliver updates to users.

The evaluation of the data we have collected, including over 150,000 published apps and associated information publicly available via Web APIs, shows TouchDevelop to be successful in various ways: apps derived from previously published apps, either by the same or different author, are more popular than apps with no version history. Most importantly, the popularity of apps created on mobile devices is on par or better than the ones created on desktop devices. Moreover, those who use TouchDevelop to author from *both* mobile and desktop devices (which is enabled by workspace replication) create the most popular apps.

The evaluation presented in this paper is based solely on the published data accessible via the TouchDevelop cloud APIs, which also are the same APIs used by the TouchDevelop IDE itself. The data analysis scripts used for our evaluation are available at http://research.microsoft.com/~moskal/touchdevelop-stats.zip.

As a running example to illustrate much of the data collected by TouchDevelop, we feature a math education app created by a TouchDevelop author (not from our team). The app is named "Quick Math" and is available via any modern browser at:

https://www.touchdevelop.com/app/#script:hhuc

Your reading of this paper will be greatly enhanced by five minutes spent exploring this app and the TouchDevelop IDE.

In the remainder of this paper, we first present background on the design of the TouchDevelop IDE and programming language (§ II) to provide context for the contributions of this paper. Section § III describes how TouchDevelop defines a general notion of publication to store a wide variety of entities in the cloud, including apps, art resources, comments, reviews, and documentation. We then describe the integration of workspace replication, version control and app store (§ IV), and present an evaluation of three years of artifacts and apps published by our user community (§ V). We finish with a discussion of related work (§ VI) and conclusions (§ VII).

II. TouchDevelop Background

This section describes the history of the TouchDevelop project and the fundamental design decisions relating to code representation, the IDE and programming language that have, for the most part, remained unchanged through the project's lifetime.

A. History

The first release of the TouchDevelop IDE was in April 2011 as a native C# Windows Phone app, six months after the project started. Publishing and version control was added in August 2011, comments and reviews were added in November 2011, and libraries in February 2012. The rewrite of the IDE as a TypeScript (a typed superset of JavaScript [3]) Web app took place in 2012 and was released in October of that year. A debugger, crash logging, and coverage collection were added in July 2013. Interactive tutorials premiered in November 2013.

B. Code Representation and IDE

In traditional IDEs, programmers edit the raw text of a program directly. TouchDevelop uses the syntax-directed editing paradigm pioneered in the late 1970's and early 1980's (as prominently featured in the Cornell Program Synthesizer [4]) to support editing of scripts using touch, with no reliance on a traditional QWERTY keyboard, physical or virtual.

TouchDevelop uses syntax-based (tree-based) editing at the level of statements, but token-based editing at the level of expressions. Such "semi-structured" editing prevents syntax errors that span several lines of code (which are difficult to fix on a tiny screen) and enables editing expressions with an adaptive on-screen keyboard at the token level, avoiding the restrictiveness of a fully structured editor (in the vein of Scratch [5]).

For expression editing, IDE keyboard buttons represent tokens, not characters, and can change dynamically based on the

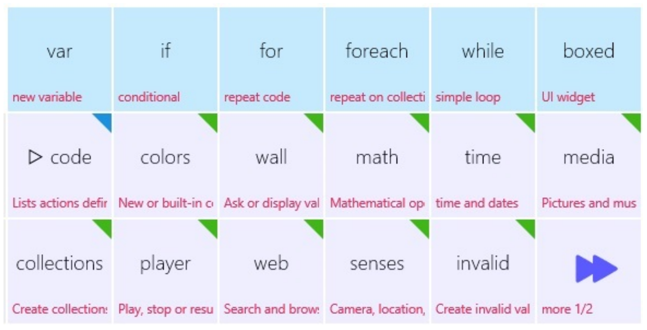
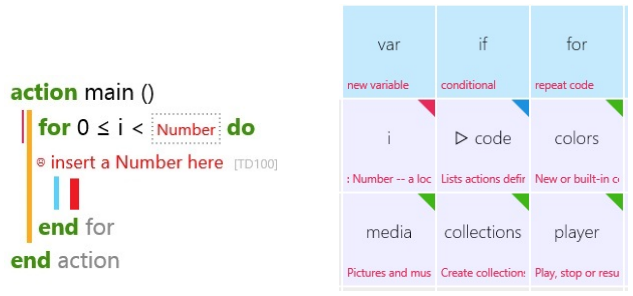


Fig. 2. Adaptive keyboard (minimal context; statement level)

.



```
action main ()
  for 0 ≤ i < Number do
    insert a Number here [TD100]
  end for
end action
```

Fig. 3. Adaptive keyboard (cursor positioned inside for loop)

.

cursor context. Experiments by colleagues in the PocketCode project show that the mix of tree-based and token-based editing is more effective than solely tree-based editing [6].

Given all the possible options and names to choose from, it is essential that TouchDevelop's onscreen keyboard adapt to the context, which generally is specified by the cursor position in the abstract syntax tree (AST), the expected type at that position in the tree, the most recently inserted tokens, and visible declarations. Figure 2 shows the keyboard when the cursor is position in an empty action (minimal context). The buttons on the top-row list the possible statements that can be inserted at this point. The second and third rows list the various namespaces in TouchDevelop's API set.

Figure 3 shows the keyboard after the addition of a "for" loop and positioning of the cursor inside the loop body. The TouchDevelop IDE creates the loop iteration variable "i" automatically; notice how the variable now appears as an option in the keyboard. The editor is aware of the context of the cursor and that the variable "i" is in scope in this context. Also notice that the IDE shows that the loop is missing an upper bound. Should we insert "i", the keyboard would change again to show operators defined on numbers.

The AST foundation underlying TouchDevelop creates numerous opportunities for improving the experience further. First, because the editing of the AST takes place via a specialized editor, the edit operations can be tracked much more precisely. For example, identity of statements can be preserved when they are moved, allowing for a more precise diff and better merge semantics [7]. Second, code can be rendered automatically, using colors, indentation, and fonts consistently. Finally, IDE plugins can easily manipulate ASTs directly, simplifying both code analysis and modification.

The basic IDE functionality (including code editing with temporary storage, compilation, running, debugging, and profiling) is fully available offline as it is implemented completely on the client side.

C. Programming Language

The simplification of the IDE alone is not sufficient to enable the convenient development of applications on mobile devices. Because screen space is limited, it is important to keep code as *simple and concise* as possible, even when it comes at the expense of some flexibility or performance. Towards this end, TouchDevelop defines an imperative language with a static type system, standard block-structured control flow constructs (such as procedures–called *actions* in TouchDevelop–loops, and conditionals), as well as built-in and user-defined types.

Instead of using separate specialized languages like HTML, CSS, or SQL, the TouchDevelop language contains special abstractions for graphical user interface [8] and replicated data types [9]. A TouchDevelop script consists of a set of top-level declarations, which can be actions, user interface pages, types, variables, art resources, as well as references to user-defined libraries (another special kind of TouchDevelop script). The interested reader can refer to the TouchDevelop book for a more detailed description of the language [10]. For the purposes of this paper, the essential points of the language are as follows:

- **Strong typing** allows us to use the type of an expression to reconfigure the adaptive keyboard to display the most likely next choices in the IDE, as previously explained.
- **APIs**: We provide numerous platform- and browser-independent APIs for accessing sensors (such as location, microphone, camera, gyroscope), media libraries (music, pictures), and Web services. This ensures that conceptually simple apps (e.g. pick a random song and play it, or take a picture and post it online) take 3 lines of code rather than 300. TouchDevelop provides over 1,500 functions in its API set.
- **User-defined Libraries**. It is impossible to anticipate all user needs in a built-in API set, so libraries can be authored in TouchDevelop itself and appear just as APIs do in the IDE.

III. Cloud Publications

Besides serving up the code of the TouchDevelop IDE to a browser, the main purpose of the TouchDevelop cloud backend is to store and serve up various kinds of *publications* created directly by authors or indirectly through various user actions (such as running an app).[2]

[2]The TouchDevelop cloud provides many other useful services such as search, RSS feeds, application export, etc.

A. Publications

Scripts are the most important among various types of *immutable* publications that TouchDevelop authors can create. Scripts contain the TouchDevelop code that makes up an app. One TouchDevelop author (not an author of this paper) created an education game entitled Quick Math, available at

https://www.touchdevelop.com/app/#script:hhuc

This overview page allows any (unauthenticated) TouchDevelop user to edit or run the script and also presents information about the script author and other script meta-data.

When this script was published (publication requires authentication), TouchDevelop assigned the permanent and unique identifier `hhuc` to this (now immutable) script. Basic information about this script is available via the Web API

http://www.touchdevelop.com/api/hhuc

which returns a JSON object. The Web API /stats [3] returns up-to-date statistics on the script's popularity: as of September 3, 2014, this script had a total of 2,216 runs and 744 total installations by 603 unique users (a user can install/uninstall an script from their device multiple times).

The publication types in TouchDevelop include:

- **scripts** are programs that can be published and shared with others. The identity of the script author and the id of the (base) script from which the new script was derived, if any, is included in the script meta-data at the moment of publication. As of September 3, 2014, the overview page of script `hhuc` shows that its base script is `yjvv` and the insight tab for `hhuc` shows that it is the base script of thirteen other published scripts (successors of `hhuc`). The up-to-date set of direct successors for script `hhuc` is available at /successors, which yields a JSON document with the relevant information.
- **comments** are free-form pieces of text attached to other publications. Comments attached to another comment form a thread of discussion. Comments can be also tagged as bugs or feature requests (/comments).
- **reviews** are much like star-ratings in an app store—they provide a quantifiable measure of the quality of a publication (*eg.*, a script, a comment, or an art resource). In TouchDevelop a user can express that they like a publication by giving it a "heart" (/reviews).
- **art resources**: Apps usually use images and sounds in addition to source code. Much like scripts, these can published as immutable entities, and have comments and reviews attached later (/art). They also can be reused by other scripts.

Associated with a script are various interesting static and dynamic metadata, such as:

- **abstract syntax tree**: via the Web API /webast, the AST of a script (as a JSON document) can be retrieved;
- **crash reports**: When a script crashes, the user is given an option to inform the author by posting a crash report (including free-form text). These reports are then automatically bucketized by stack trace and provided in the insights tab of a script's overview page. The script crash reports are available via the /runbuckets API.
- **profile and coverage information**: Some script runs are randomly selected for profiling or coverage collection. All such information is automatically aggregated and attached to the script for everyone to see via the "insights" tab in the overview page for the script. The raw coverage data is available via the /coverage API.

TouchDevelop supports several special kinds of scripts:

- **libraries** are scripts that can be used as components by other scripts; libraries implicitly define an abstract interface that hides details of the library implementation from the library consumer;
- **documents** are a special kind of script that use the popular markdown text formatting language enhanced with constructs to refer to script elements to provide a form of literate programming. Document scripts are rendered specially by TouchDevelop. For an example, see the documentation on TouchDevelop's "move to library" refactoring.
- **interactive tutorials** build on top of document scripts and are supported by a tutorial engine that gently guides an author-to-be through the process of creating a script with the IDE (for an example, see https://www.touchdevelop.com/pydja).
- **plugins** are scripts that let TouchDevelop authors extend the IDE. Plugins can be invoked from various points in the IDE and operate on the AST of the current script. They run locally in the IDE, but can invoke Web services to do the actual work (§ III-C gives more details).
- **forums**: Comment sections of a few designated empty scripts serve as general forums for the users. As elsewhere, nested comments can be attached forming discussion threads.

B. Ranking of scripts and authors

Scripts are ranked according to the number of hearts they receive, and the number of times they are installed and run. This score decays over time to let newer scripts rise to the top. Additionally, authors are assigned a score based on the number of hearts their publications receive, the number of followers they have, the number of different features of TouchDevelop they have used, and the number of days they have been active. The score is displayed prominently in the IDE and we have anecdotal evidence of authors taking their score very seriously.

C. Openness and plug-ins

Scripts are in the public domain from the point of publication on and can be freely forked by any TouchDevelop user. As we have demonstrated above, all TouchDevelop publications (scripts, comments, art resources, crashes, etc.) are available via Web APIs. The full set of Web APIs is documented at:

[3] From now on, when we reference a TouchDevelop Web API specific to the script `hhuc` we'll omit the prefix `http://www.touchdevelop.com/api/hhuc`.

https://www.touchdevelop.com/help/cloudservices

We emphasize that the TouchDevelop IDE itself uses these APIs.

Plugins are authored as TouchDevelop scripts, avoiding security issues with foreign JavaScript code running under the domain of TouchDevelop and dramatically lowering the barrier to entry for plugin authors. For example, a script to rename top-level declarations while updating the references (*eg.*, to enforce a naming convention) is six statements long (script /rmaf). Additionally, since plugins also are scripts, they can be forked, reviewed, and commented on as usual.

Technically, the TouchDevelop AST maps to a JSON object, which can be inspected and modified using regular JSON APIs. This enables processing of the data on a different server, not affiliated with TouchDevelop. Such cloud plugins can alleviate problems with limited computational capabilities of the phone, for example to run various static analysis of the code. As an example, researchers at ETH developed a static analysis plugin backed by a server in Zurich [11].

IV. Integrating Workspace, Version Control and App Store

This section describes how TouchDevelop integrates replicated workspaces with version control and app publication. In this section, we drop the distinction between "authors" and "users" of TouchDevelop scripts and refer to them just as "users".

A. Replicated Workspace

Users no longer rely on a single device for running applications. Often, they have multiple PCs and/or laptops (at work and/or at home), they carry a smartphone at all times, and they use a tablet on the go or in recreational spaces. Thus, they are likely to frequently switch between devices, or even use multiple devices at the same time. As they do, they expect continuity. In particular, they expect that their data and preferences (their personal *workspace*) are synchronized between devices, and backed up in the cloud so that they remain continuously available even if some or all of the devices are powered off.

TouchDevelop provides a *workspace* in its cloud for each authenticated user The workspace maintains a set of scripts and this set is replicated on any device the user signs into TouchDevelop from, ensuring that the user has access to their most recent (private) changes to scripts. The state of the workspace is automatically synchronized between different devices of the same user.

In more detail, a workspace consists of a set of "slots". Each slot is identified by a unique GUID, and contains mutable fields for: a version identifier; the script contents; other metadata about the script. Each user can have multiple workspaces — typically, one per device or browser. Each user also has a (single) persisted workspace in the cloud. Every time an authenticated user connects to the cloud via the TouchDevelop client, the latest version of each slot is synced from or to the cloud.

Each workspace features a per-slot history feature which periodically snapshots all modified slots in the workspace and saves the history to the cloud, so that the user can back up in time regardless of which device they are working from. The workspace history feature is separate from the version control system and private to each user. Both synchronization and history are common features of modern cloud file storage services (*eg.*, DropBox) and are thus well-understood by users.

B. Version Control and Dependency Management

The TouchDevelop version control system stores a forest of immutable scripts, where each script has an associated user id (the authenticated user who published the script), and the base script (the script's parent in the tree) from which it was derived. A script with no base is called a *root script*. The *base chain* of a script S is the sequence of scripts from S to the root script of its tree (via the parent pointers). The IDE makes it easy for users to navigate up the base chain of a script to visit its tree ancestors (via the overview page).

A published script S from the TouchDevelop cloud is installed into a user's workspace when a user edits or runs S; more specifically, script installation creates a private fork of script S (recall that published scripts are immutable) in the user's workspace that can be modifed and ultimately published as a new script S', derived from S. When a user publishes script S', TouchDevelop records that script S is the base of script S'.

The immutability of scripts plays a critical role in **dependency management**. Just before script S' is published, all its dependencies (other scripts used by S', as libraries, and art resources used by S') are automatically published as well (if they have been modified) and their resulting identifiers are stored in S'. This way, every script has a *transitive consistent snapshot* of its dependencies. The dependencies of a script are defined by the static semantics of the TouchDevelop language and exposed via the TouchDevelop compiler to the version control system. This makes it impossible to create dangling references.

Stated another way, if a script compiles successfully and then is immediately published to the TouchDevelop cloud (with script id I) then any user that installs the script with id I from the TouchDevelop cloud will find that it compiles successfully. However, it is possible for a user to publish a script that does not compile successfully. As TouchDevelop is a vehicle for learning programming, we opt for maximal sharing, even of ill-formed scripts.

Given that we make it easy for a user A to publish buggy script B, a very common scenario is for a user other than A make a bug fix to B, which user A would then like to inspect and potentially merge back into B. Rather than create a whole new mechanism for patches, we simply require that users share updates via script publication mechanism and provide a new (3-way) AST-based merge operation.

The merge functionality is exposed in TouchDevelop as follows: every published script has a merge button associated with it; when a user clicks the merge button for script

S, TouchDevelop presents the set of scripts in the user's workspace that have a (least) common ancestor $Base$ with S in the version control tree. The user selects a script T from this set and TouchDevelop performs a 3-way merge of $(Base, S, T)$ into script T. Most often, it will be the case that $Base = S$ (as in the common scenario of the previous paragraph).

To support a precise 3-way merge operation, the TouchDevelop IDE generates unique identifiers for each AST node (hidden from the user) that are stored in the script. Furthermore, when a merge operation takes place, a record of the merge operation is inserted into the meta-data of the target script so that duplicate merges can be prevented and the provenance of the target script can be reconstructed after its publication.

C. App Store and Updates

The TouchDevelop app store layers a different notion of "version" on top of the version control system for the purposes of identifying the preferred version of a script, which will be displayed in TouchDevelop as an "app" for others to download.

A TouchDevelop app is identified (named) by a primary key consisting of a script's name N and a user id I. The secondary keys are script id S (whose' associated name and user id are N and I, respectively) and a timestamp T (the time at which S was published by user I). Among all the tuples (N, I, S, T) with primary key (N, I), TouchDevelop chooses the S with the most recent timestamp T as the "current version" of the app (N, I) to feature.

Thus, if user I updates any script with the primary key (N, I) and publishes it as S' then S' will become the "current" app. This allows a user to easily revert an app to a previously published script by republishing it.

Any user who has an (unmodified) published script with name N and user id I in their workspace receives a notification (in the IDE) that there is an update available. The user can choose whether or not they want to update.

Some of the meta-data associated with a script publication (such as comments, reviews, etc) is aggregated by TouchDevelop and presented for the "current" app in the overview page of its associated script. Other information, like crash dumps are specific to a version of a script and are not aggregated or presented in the "current" app, but only in the overview page of the associated script.

V. Evaluation

This section presents an evaluation of TouchDevelop and the two feedback cycles from Figure 1. § V-A discusses the IDE itself and interaction of the authors of this paper with app authors and users, while the remaining subsections focus on app authors, their creations, and interactions with app users.

A. TouchDevelop itself

TouchDevelop consists of a cloud back-end comprising about 100KLOC of C# running in Windows Azure, and a client running in HTML5 Web browsers. The IDE part of the client (AST operations, compiler, editor, debugger etc.) is implemented in about 75KLOC of TypeScript; the runtime libraries are 70KLOC.

Every source code check-in to the client is built automatically and uploaded to the cloud back-end generating a uniquely named release. At any time there is one release labeled "current" and (a possibly different) one named "beta". The "current" release is moved about every other week, and the "beta" is moved several times per week. Users with a high score are encouraged to try the beta version.

In case of an unexpected exception or assertion failure in the client, we log a crash report in the cloud. The client also streams instrumentation telemetry data to the cloud. As of September 2014, we collect about 200 crash reports and 35,000 telemetry reports per day, from about 1500 users.

We found the crash reports to be tremendously useful. The cloud back-end automatically bucketizes crash reports by stack trace and type, and sorts them by number of occurrences. This lets us focus on the crashes occurring frequently and impacting more users. Crash reports include the current script being edited and 1000 recent log messages and instrumentation events. The log is particularly useful in the presence of async APIs in JavaScript, which often make stack traces non-informative.

The analysis of instrumentation data is simpler: it can be visualized over time and categorized by different kinds of devices. We have used it on a number of occasions when deciding to remove unused fragments of user interface code from the IDE.

In addition to automated tracking, we also let users, particularly the ones using the "beta" release, report bugs in a dedicated forum. These then can be categorized, assigned, and tracked. Given our limited resources, this form of crowd-sourced testing enabled by the large TouchDevelop user-base has proven very useful.

B. Scripts and users

This section provides basic data about the users of TouchDevelop, the scripts they publish, and growth trends. The analysis is based on the public data about users and their published scripts (see § III-C).

The data-set contains 164,267 scripts published between August 2011, when we first introduced script publishing, and September 2014. We have excluded from further analysis 7,586 scripts published from various system and testing accounts, leaving us with 156,681 scripts.

A script *feature* is a built-in function name, a qualified library function name, or a language feature name (*eg.*, "if-statement", "object type definition" or "assignment").

A *feature multiset* for a script contains each feature as many times as it is used in a given script. In particular, the multiset does not contain literals or art references, which are customizable in automatic tutorials, and are commonly changed during *rebrandings*—when a user forks a script and only changes a bit of text or a picture.

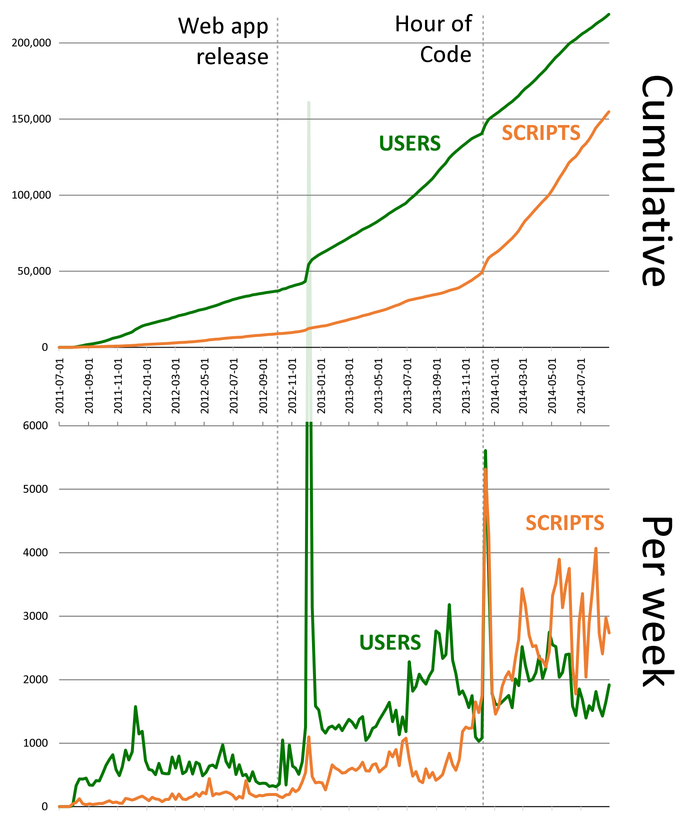


Fig. 4. Growth of TouchDevelop

Trivial scripts are ones which have no base script and share the exact feature multiset with at least ten other scripts. These are either very small, or are the result of completing a tutorial.

Overall, 48% (75,957) of all scripts are non-trivial. There are 220,783 users of whom 44,956 have published at least one script, and 19,328 have published at least one non-trivial script.

Figure 4 shows the growth of TouchDevelop over time—published scripts and registered users, both cumulative and per-week. User registration is optional since October 2013: we estimate the number of unregistered users to be around 500,000. There was a significant bump in number of users (if not scripts) following a round of publicity after the initial release of the Web IDE. Similarly, the Hour of Code event in late 2013 brought in quite a few new (but trivial) scripts.

Figure 5 shows the size of published scripts. While the majority of scripts are small (their median size is 24 statements), 18,135 scripts contain more than 100 statements and 1,559 scripts contain more than 1000 statements. Scripts above 5000 statements are outliers—there are only 132 (0.1%) of them—and the biggest script has 9282 statements. Scripts which are successful, measured by the number of runs, tend to be larger— scripts with over 100 runs have a median size of 71 statements and scripts with over 500 runs have a median size of 90 statements.

In the remaining sections we examine how various factors influence users' success in developing apps. We use the *popularity* of a published script, defined as the number of runs of the script, as a proxy for its quality and thus user success. This decision is discussed further in the Threats section (§ V-F).

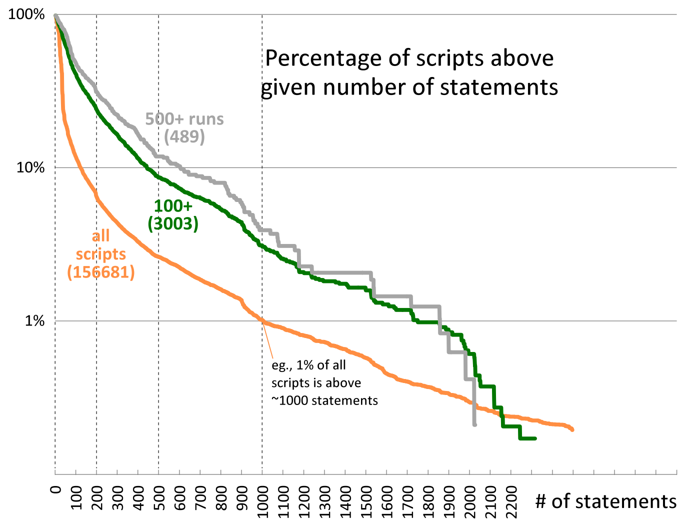


Fig. 5. Distribution of script sizes

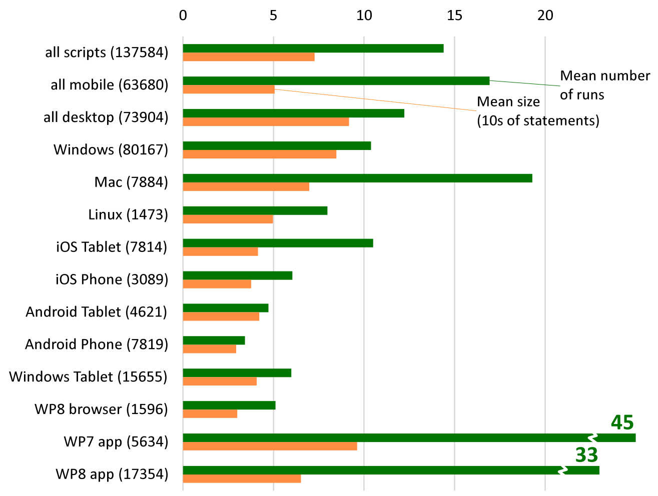


Fig. 6. Script popularity by publication platform

C. Effects of replicated workspace

The main benefit of having the workspace automatically replicated is the ability to seamlessly switch between development on different devices, in particular mobile ones. We thus explored correlations between publication devices and script popularity.

Figure 6 shows the mean number of runs and the mean size for scripts published from different kinds of devices. Overall, scripts published from mobile devices (phones, tablets, music players, etc.) are smaller, yet more popular than ones published from desktops (keyboard-equipped desktop and laptop computers). This suggests people are putting more effort into programming from a mobile device.

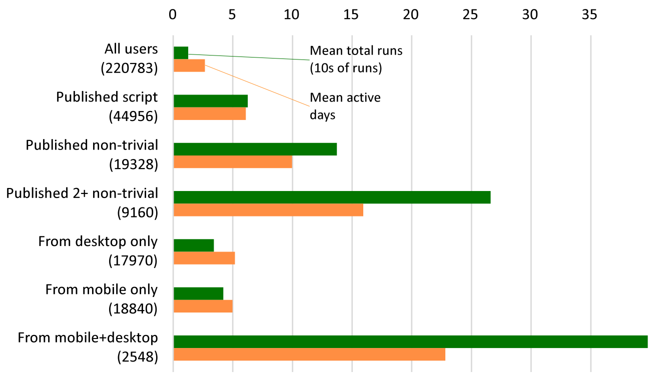


Fig. 7. Script popularity by user type

The good showing of mobile platforms is mainly due to the dedicated TouchDevelop apps for Windows Phone. Our Android app is relatively recent and not yet as feature complete. Tablets are generally also showing good results, comparable with desktop machines.

We did not collect data about publication device at the beginning—the plot shows data for the 88% of scripts for which we did collect publication device. Also, Windows tablets are a bit difficult to categorize as they may or may not sport a keyboard, which may or may not be used. They are thus included in both Windows numbers and separately.

Figure 7 shows the total number of runs of all scripts published, averaged over given user set (and divided by 10 to fit in the plot). It also shows the mean number of days the user was active.

Similarly to the data for scripts, users publish higher quality scripts from mobile platforms. In both cases they use the platform for about five days on average. However, users who publish from both desktop and mobile platforms produce much better scripts and use TouchDevelop for almost a month on average.

The last point is particularly encouraging, as this kind of usage is directly enabled by the replicated workspace.

D. *Effects of version control*

TouchDevelop version control and dependency tracking allows users to create updates of their existing scripts, fork and modify scripts as well as utilize libraries. Here we discuss effects of these activities on script popularity.

We use the term **remix** for a script with more than one author in its base chain. A *direct remix* is a script authored by a user different from the base script's author.

The *update size* is the cardinality of the multiset of features used in given script minus the feature multiset of its base (if any). The cardinality of the sum of all update sizes, which is a measure of published edit operations, is similar for trivial and non-trivial scripts (8.3M and 7.1M respectively). On the other hand, there are 1.8M statements in trivial scripts (which have no base) and 9.8M in non-trivial ones, suggesting that an average non-trivial statement was republished unchanged about 6 times.

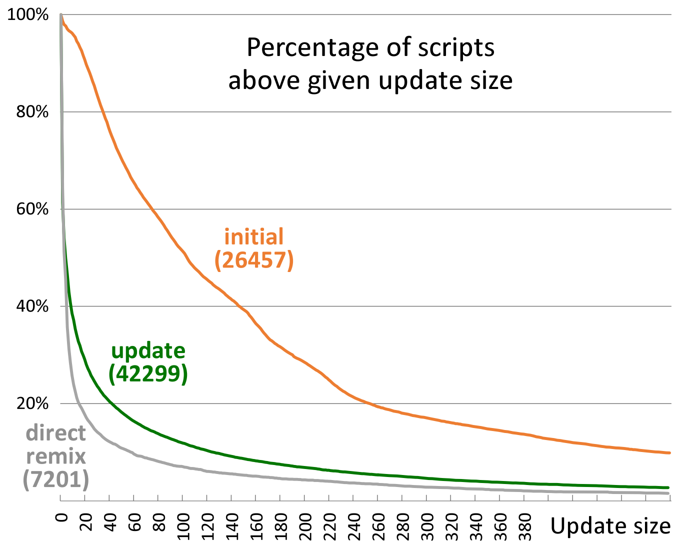


Fig. 8. Distribution of update sizes

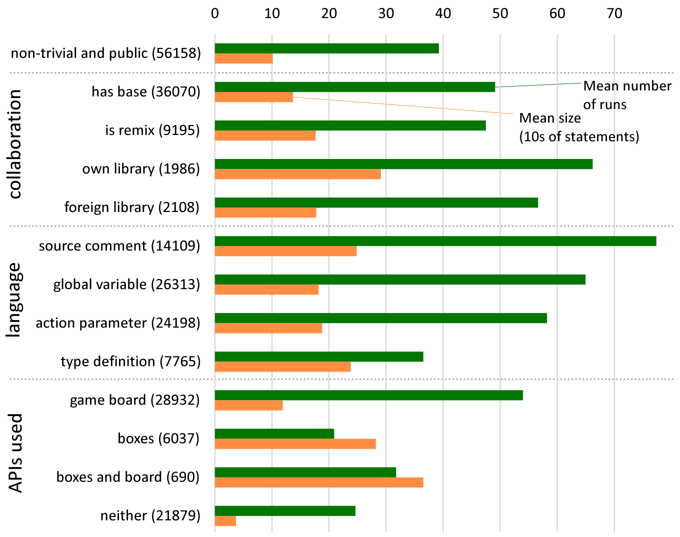


Fig. 9. Script popularity by feature used

Figure 8 shows the update sizes for scripts with no base (*ie.*, initial publications), scripts where the base has the same author (*ie.*, updates), and scripts where the base has a different author (*ie.*, direct remixes). The data is for non-trivial scripts only. The initial publication is by far the biggest, with small incremental updates after that. Remixes are less numerous and on average 40% smaller than updates.

Typically, there are at most a few updates published for a given script, however longer update sequences occur: the maximal length of a base chain is 314, with 995 scripts over 50, 8,352 over 10, and 49,500 over 1.

Figure 9 shows script popularity for scripts using different features of the platform or the language. The data is limited to non-trivial and non-hidden scripts. [4] Hidden scripts account for about 15% of all scripts, but are more common in some of the buckets we analyze (though not in the per-platform buckets from previous sections). Hidden scripts are hardly run by anyone other than their author, so they get artificially low scores.

Scripts which have a base are more popular and bigger than average. Remixes are bigger yet and of slightly lower popularity. Additionally, over half of the scripts have a base, and almost a fifth are remixes.

Libraries as a form of modularization (*ie.*, the author of library and the script is the same) seem very successful. Usage of libraries published by others also gives a boost to popularity. We have excluded usage of libraries published by our system accounts, which are automatically included in templates.

The positive effects of re-publishing of scripts and libraries point to the usefulness of our version control and dependence tracking scheme. The high number of scripts leveraging these features point to their simplicity.

E. Effects of language features

The second set in Figure 9 focuses on the usage of different language features. Scripts with comments in the source seem more popular (possibly because they are utilized by more advanced users). Other somewhat advanced features (global variables and functions with arguments) also have positive influence on popularity. However, users do not seem to know how to handle more advanced features, like type definitions and boxes [8] (see below).

The last set splits the scripts by the main API used. Scripts that use the game board (which includes physics and 2D graphics engines) are most numerous and popular. Boxes are TouchDevelop's way of constructing more complex UIs. They are an advanced feature, which users seem to find hard to master. Finally, it is possible to create very simple scripts (note the low average size) with simple UI, which use neither game board or boxes.

The relative run counts of games and apps may be due to users being more inclined to play games than use productivity apps. However, even the gap between apps with and without boxes is significant.

F. Threats To Validity

It is important to recall the context of the TouchDevelop project and the threats to validity that may prevent our results from generalizing to other CIDEs,

Most of TouchDevelop users are beginning programmers. However, there is a small but very influential group of more experienced developers who produce good libraries and base scripts for remixes by the larger group. This kind of division is a natural one in many software development scenarios, but the experience gap between the two groups is likely to be much larger in TouchDevelop. Moreover, the motivation structure in TouchDevelop is vastly different, with most users just checking it out for fun. Furthermore, most of the scripts are small and mobile-specific, and are written in a custom-designed programming language designed for syntax-directed editing.

We have used number of runs of a published script as a measure of success. Our assumption is that once a user manages to develop an app, they will publish it and it will get run by the author and other users. Of course, the user may not publish, or other users may fail to find (and thus run) the script. However, the number and quality of publications does not suggests users are particularly shy about publishing. We also observe lots of scripts being run by non-authors.

Alternatively, we could use the number of installations of a script or the number of hearts (positive reviews) given to a script. We found these to be correlated, but the number of runs is the most informative, especially for the majority of scripts, which do not have any hearts, but are run a few times.

Profile, coverage, and crash logs also could be used as measures of quality and popularity, but they are often missing and do not capture if the script is performing useful functions.

VI. Related Work

Many aspects of TouchDevelop can be found in lesser combination in existing projects and websites; to the best of our knowledge, the deep integration of an IDE, replicated workspaces, a version control system, and app store is unique to the TouchDevelop project.

A. Software Configuration Management

The primary contribution of this paper is the novel combination of ideas from software configuration management [12] in a new domain (a cloud-based IDE for creating mobile apps).

Workspaces. Most traditional IDEs, such as Eclipse and Visual Studio, support a notion of a workspace local to the developer's machine, but do not support workspace replication across devices. As we have seen, workspace replication allows users to develop and maintain TouchDevelop scripts from any device, including the mobile device they intend their app to be delivered on. Our evaluation shows that the quality of apps is significantly affected by the set of devices the TouchDevelop scripts were authored with.

Version Control. On the other hand, TouchDevelop's (centralized) version control system (VCS) is highly simplified compared to traditional VCSs: TouchDevelop's VCS maintains a forest of immutable publications, which makes it easy for users to fork a new version of an existing publication. TouchDevelop's VCS is not file- or directory- based: TouchDevelop publications are stored in the cloud and live in a flat namespace of unique identifiers (such as `hhuc`) - the publications are discoverable via search, a tagging mechanism, and ranking scores, as well as via other meta-data associated with publications (i.e., whether or not they are libraries, documents, tutorials, etc).

TouchDevelop has no notion of "check-out" or "check-in" operations associated with many VCSs. Instead, the action of

[4]Hidden scripts are published and publically accessible via the Web APIs but are not indexed by TouchDevelop's search engine.

editing a published script creates a fork of that script, which later can be published. An explicit merge operation is used to combine the contents of a published script with a script in a user's workspace. These three operations (edit, publish, merge) are sufficient for TouchDevelop programmers to share and collaborate with one another, which is much simpler than popular but complex VCSs such as Git [13].

Dependency Management. TouchDevelop layers other features of a VCS on top of its tree of immutable publications via meta-data imbedded in the publications: a new publication P's dependencies are published (and thus become immutable) and their version identifiers embedded in P before it is published. This feature is a form of "strongly typed version control", as identified by Perry [14], which requires an interface between the version control system and the compiler. The use of explicit dependences between software artifacts to optimize and incrementalize the build process has been well studied [15], [16]. We make use of the same idea to make it easier for our users to deploy and update their apps.

B. Web-based Software Development and CIDEs

Community websites such as StackOverflow demonstrate how to encourage programmers to share knowledge, and online repositories such as GitHub [2] combine cloud storage, social functions to encourage collaboration, outsourcing functions like build to other web services. However, neither one supports both the development activity (IDE) and the deployment of applications (app store). Recent work by Ponzanelli and colleagues integrates StackOverflow into the Eclipse IDE [17].

Providing the functionality of traditional IDEs in a Web site is becoming increasingly popular, and there are many web-based IDEs available today, such as codenvy.com, koding.com, or visualstudio.com. While these sites provide typical IDE functionality (develop, compile, test, debug) and parts of TouchDevelop experience (ubiquitous workspace and code repository), they fall short on the many other aspects of our vision.

There is no deep integration of social features to support the online community and foster open collaboration. Also, support for tracking the whole life cycle of applications (including deployment and telemetry) is only slowly emerging, and typically limited to the parts of the application that execute in the cloud. Few web-based IDEs support offline operation, as TouchDevelop does.

Finally, as far as we have seen, web-based IDEs all but ignore the mobile experience and still squarely focus on programming from the traditional desktop with large screens, keyboards, and text-based programming languages.

C. Development for Non-expert Programmers on Mobile

PocketCode [18] is a tool similar to the initial version of TouchDevelop—it focuses on single-user programming directly on a phone or tablet. The programming language is far simpler and its main aim is education.

AppInventor [19] lets users create Android apps by dragging and dropping blocks representing various ASTs on a separate PC. The programming language is simpler than in TouchDevelop (for example lacking library abstractions), and the cloud and social support is more limited. On the other hand, the support for Android-specific APIs and UI is superior in AppInventor. Interestingly, the code editor is fully structured—expressions are also edited as trees. The effectiveness of the two approaches was recently compared [6].

VII. Summary

We have presented a publication model for scripts and associated meta-data, ranging from comments and reviews to run-time coverage information and stack dumps, as well as an integration of workspace replication, version control, and app store. Our evaluation of the TouchDevelop CIDE shows that this combination has been effective in attracting a vibrant user community that shares and remixes scripts.

Since the TouchDevelop CIDE has complete knowledge of all the code and libraries needed to run an app (via version control + automated dependency tracking) it is very easy for anybody to fork a published application to make modifications. This enables spontaneous collaboration and encourages users to openly exchange code.

A CIDE integrates not only the development and testing of an application, but the whole lifecycle, which also includes publishing of the application in the app store, and the publishing of updates. Thus, a CIDE can automatically collect telemetry information and report it to the author. For instance, it can report how many users have run a script, it can display detailed code coverage, and it can report crashes. It also collects user ratings, comments and suggestions.

Last but not least, the use of a CIDE simplifies the collection of data across the whole lifecycle of an application, and allows us to quickly incorporate lessons learnt. In TouchDevelop, we make all versions of all applications and variations (by all authors) publicly available for research, accessible via the Web (see § III-C).

References

[1] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich, "TouchDevelop: programming cloud-connected mobile devices via touchscreen," in *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, ser. ONWARD '11, 2011, pp. 49–60.

[2] "GitHub," http://github.com.

[3] "TypeScript Langauge Website," http://www.typescriptlang.org/.

[4] T. Teitelbaum and T. Reps, "The cornell program synthesizer: A syntax-directed programming environment," *Commun. ACM*, vol. 24, no. 9, pp. 563–573, Sep. 1981. [Online]. Available: http://doi.acm.org/10.1145/358746.358755

[5] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch programming language and environment," *Trans. Comput. Educ.*, vol. 10, pp. 16:1–16:15, November 2010.

[6] A. Harzl, V. Krnjic, F. Schreiner, and W. Slany, "Comparing purely visual with hybrid visual/textual manipulation of complex formula on smartphones," in *DMS*, 2013, pp. 198–201.

[7] S. Apel, O. Leßenich, and C. Lengauer, "Structured merge with auto-tuning: balancing precision and performance," in *ASE*, 2012, pp. 120–129.

[8] S. Burckhardt, M. Fähndrich, P. de Halleux, S. McDirmid, M. Moskal, N. Tillmann, and J. Kato, "It's alive! continuous feedback in UI programming," in *PLDI*, 2013, pp. 95–104.

[9] S. Burckhardt, M. Fähndrich, D. Leijen, and B. Wood, "Cloud types for eventual consistency," in *European Conference on Object-Oriented Programming (ECOOP)*, ser. LNCS, vol. 7313. Springer, 2012, pp. 283–307.

[10] N. Horspool and N. Tillmann, *TouchDevelop - Programming on the Go*. Apress, 2013. [Online]. Available: https://www.touchdevelop.com/docs/book

[11] L. Brutschy, P. Ferrara, and P. Müller, "Static analysis for independent app developers," in *Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, 2014, to appear.

[12] J. Estublier, D. Leblang, A. v. d. Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber, "Impact of software engineering research on the practice of software configuration management," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 4, pp. 383–430, Oct. 2005.

[13] S. Perez De Rosso and D. Jackson, "What's wrong with git?: A conceptual design analysis," in *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming &#38; Software*, ser. Onward! '13, 2013, pp. 37–52.

[14] D. E. Perry, "Version control in the inscape environment," in *Proceedings of the 9th International Conference on Software Engineering*, ser. ICSE '87, 1987, pp. 142–149.

[15] W. F. Tichy, "Smart recompilation," *ACM Trans. Program. Lang. Syst.*, vol. 8, no. 3, pp. 273–291, Jun. 1986.

[16] Z. Shao and A. W. Appel, "Smartest recompilation," in *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '93, 1993, pp. 439–450.

[17] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 1295–1298.

[18] W. Slany, "A mobile visual programming system for Android smartphones and tablets," in *VL/HCC*, 2012, pp. 265–266.

[19] J. Liu, C.-H. Lin, P. Potter, E. P. Hasson, Z. D. Barnett, and M. Singleton, "Going mobile with App Inventor for Android: a one-week computing workshop for k-12 teachers," in *SIGCSE*, 2013, pp. 433–438.