Porta: Profiling Software Tutorials Using Operating-System-Wide Activity Tracing

Alok Mysore
UC San Diego
La Jolla, CA, USA
amysore@eng.ucsd.edu

Philip J. Guo
UC San Diego
La Jolla, CA, USA
pg@ucsd.edu

ABSTRACT

It can be hard for tutorial creators to get fine-grained feedback about how learners are actually stepping through their tutorials and which parts lead to the most struggle. To provide such feedback for technical software tutorials, we introduce the idea of *tutorial profiling*, which is inspired by software code profiling. We prototyped this idea in a system called Porta that automatically tracks how users navigate through a tutorial webpage and what actions they take on their computer such as running shell commands, invoking compilers, and logging into remote servers. Porta surfaces this trace data in the form of profiling visualizations that augment the tutorial with heatmaps of activity hotspots and markers that expand to show event details, error messages, and embedded screencast videos of user actions. We found through a user study of 3 tutorial creators and 12 students who followed their tutorials that Porta enabled both the tutorial creators and the students to provide more specific, targeted, and actionable feedback about how to improve these tutorials. Porta opens up possibilities for performing user testing of technical documentation in a more systematic and scalable way.

Author Keywords

tutorial profiling; activity tracing; software tutorials

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

It can be hard for experts in any field to write high-quality documentation, instructional materials, and step-by-step tutorials since they are too intimately familiar with their own subject matter and thus cannot easily predict where new users might struggle [22, 26]. Despite their best efforts, it is all too easy to omit certain tutorial steps, gloss over subtle details, or provide incomplete explanations. This is an instance of the *expert blind spot* effect [27] where experts have trouble relating to what novices might know and not know.

UIST '18, October 14–17, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5948-1/18/10...$15.00
https://doi.org/10.1145/3242587.3242633

Even worse, since tutorials are static documents or videos, once they are released, their creators cannot easily get a sense of how people navigate through them or what parts they frequently get stuck on. Thus, learners inevitably struggle in ways that tutorial creators are not able to foresee. Some ask for help in discussion forums or mailing lists, but those resources are disconnected from the context of the original tutorials. These limitations inspired our core research question: ***Can we provide effective feedback to tutorial creators about how learners are actually stepping through their tutorials and which parts lead to the most struggle?***

In this paper, we address this question for technical software tutorials, which are commonly found on the websites of open-source projects, in blog posts, and in instructional materials of programming classes and MOOCs. These tutorials are especially challenging for their creators to test and maintain due to the complexities of software environments: 1) If they create a tutorial that works perfectly on their own computer, chances are that it will not work properly on some other users' computers since they may have subtly differing versions of libraries or configuration settings that are hard to test for. 2) Even if they make a perfectly robust tutorial, some parts will inevitably break over time as users' installed versions of libraries, frameworks, and operating systems get upgraded.

To address these challenges of tutorial testing and upkeep, we introduce the idea of *tutorial profiling* and demonstrate an automated approach to doing so based on operating-system-wide activity tracing. Our idea is loosely inspired by code profiling [16, 31], which instruments and runs source code in order to surface performance and quality problems. We instantiated our profiling approach in a prototype called Porta[1] for macOS. Figure 1 illustrates an example use case:

1. Imagine you are the creator of a tutorial on how to make a full-stack web application with the Angular framework and TypeScript compiler for the frontend [30], Python's Flask framework [2] for the backend, and deployment to a Linux-based hosting provider. Your tutorial consists of a webpage with step-by-step textual instructions, example command and code snippets, and embedded screencast videos.

2. To get feedback on your tutorial, you find test participants by recruiting either in person or remotely. Each participant installs the Porta macOS app and accompanying Chrome

[1] **P**rofiling **O**perating-system **R**ecordings for **T**utorial **A**ssessment

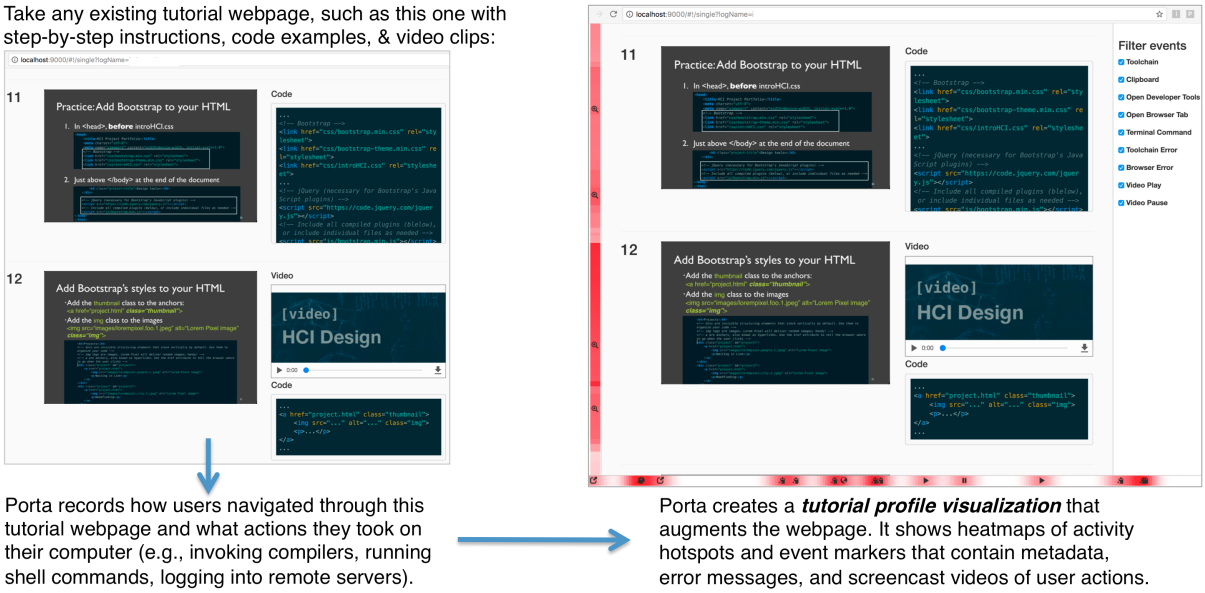

Figure 1. Porta provides feedback to tutorial creators about how users navigated through their tutorials and what application errors they encountered.

browser extension. Then they activate Porta and start reading through your tutorial webpages in their web browser.

3. As each participant follows your tutorial step by step, they launch a variety of applications to accomplish the indicated tasks. Here they might open: a) a text editor to write code, b) a terminal to run Python commands for the Flask backend, c) a terminal to run TypeScript build commands for the frontend, d) a terminal to ssh into a Linux server to run online deployment commands, e) a web browser window to configure the hosting provider's settings, f) another open browser window to debug the app they are building.
4. Porta records a screencast video of the session and automatically tracks selected activities within all of these applications to build up a cross-application *usage profile*. This profile includes the code that was compiled and the commands that executed in both local and remote shells, along with what error messages resulted from those actions.
5. Porta also tracks user navigation within tutorial webpages, such as mouse and scroll positions, seek positions of embedded video players, and what other tabs were opened.
6. Porta combines webpage navigation and usage profile data to create a *tutorial profile visualization* that shows what users were doing on their computer and what errors they encountered as they were viewing each part of the tutorial. This visualization augments the tutorial with heatmaps of activity hotspots and markers that expand to show event details, error messages, and screencast videos of user actions.
7. By inspecting this profiler output, you can see where participants had trouble or what unexpected actions each one took. You can use these insights to revise your tutorial or show it to participants in post-study debriefing interviews to help them reflect on why they struggled at specific steps.

Porta works on existing unmodified webpages viewed in Chrome. It is best-suited for profiling technical tutorials in domains such as software development, data science, system administration, DevOps, and computational research. What these have in common is that they involve a heterogeneous mix of command-line and GUI applications. Since users have subtly differing software setups on their computers, it is hard for tutorial creators to take this vast array of possible configurations into account when writing their instructions [26].

The potential significance of Porta lies in how it opens up opportunities for ***user testing of technical documentation in a more systematic and scalable way***, which can in turn help make software easier to learn and use.

Beyond our Porta prototype, our overall approach of combining data on how users navigate through tutorials with the actions they take while following them could generalize to other technical domains such as hardware, 3D fabrication, or crafting. To extend to those domains, one must build the proper monitoring probes and domain-specific data visualizations.

To assess Porta's efficacy, we used it to perform user studies on 3 tutorial creators and 12 learners who tried to follow their tutorials. We recruited 3 instructors who created tutorials for their courses on Python, Git, and web design, respectively. We had 4 undergraduate students try to follow each tutorial while their actions were recorded by Porta. When we showed Porta's output to the students, we found that it allowed them to provide more specific and targeted feedback about how to improve those tutorials. Finally, when we showed Porta's output back to the 3 instructors, we found that it allowed them to discover surprising insights about student behavior and come up with actionable ideas for improving their own tutorials.

The contributions of this paper are:

- The idea of *tutorial profiling*, which is inspired by ideas from code profiling [16, 31] and activity visualization [19].
- Porta, a prototype that instantiates this idea for macOS. Porta consists of an OS-wide activity recorder, a webpage navigation tracker, and interactive profiling visualizations.

RELATED WORK

Porta introduces the idea of tutorial profiling, which is inspired by work in application usage profiling, information scent, improving web tutorials, and OS-wide activity tracing.

Application-Specific Usage Profiling

Porta extends the long lineage of systems that visualize *usage profiles* of how people use specific software applications.

Edit Wear and Read Wear [19] pioneered the use of graphical marks on the scrollbar of a text editor to show where users were most frequently editing or reading a document, respectively. Follow-up work showed visualizations to help users re-find parts that they had previously visited [7]. Patina [24] is an application-independent generalization of these ideas; it displays heatmaps over UI elements on Windows applications to show which ones were frequently accessed. Our Porta system displays a form of read wear on tutorial webpages in the browser, which is powered by traces of how users interacted with a multitude of *other* applications on their computers.

In a similar vein, LectureScape [20] collects usage profiles of how people interact with educational videos. It displays a histogram-like visualization overlaid on the scrubber timeline. Peaks in this visualization represent where lots of viewers either paused or navigated to, which could indicate potential points of high interest or confusion. Porta also tracks video navigation but additionally correlates that data with how users interact with other applications on their computer.

ERICA [12] and ZIPT [11] capture usage profiles of how users navigate through Android mobile apps. It displays flow visualizations to help UX designers see which parts of the app their users get stuck on. Porta is motivated by similar goals in how it displays visualizations to show where users struggle when following web-based software tutorials.

Code coverage and profiling tools [16, 31] now exist for most major programming languages. These tools show how many times each line of code (or function call) ran and how much time it took. Theseus [23] improves on this idea by continually running user code and displaying always-on profile visualizations in the margins of the code editor. Bifröst [25] further extends code profiling to mixed hardware-software systems. In contrast, Porta is a tutorial profiler that tracks how a user "runs" a software tutorial by following it step-by-step and invoking various applications on their computer.

Porta innovates upon this class of prior work by introducing the idea of building usage profiles for *software tutorials* rather than for specific pieces of software. It is designed to profile tutorials whose instructions span multiple applications including web browsers, IDEs, terminals, and commands executed on remote servers. This makes it well-suited for technical tutorials in domains such as software development, computational research, and system administration.

Multi-Application Information Scent

The profiling sidebar that Porta displays over webpages provides information scent [28] that helps tutorial creators hone in on which parts their users struggled with. It was inspired by systems that connect information scent across applications.

Codetrail [15] and HyperSource [18] augment an IDE by connecting it to a web browser to link code and documentation. Codetrail enables interactions such as automatically opening documentation related to code that the user is currently editing. HyperSource automatically associates code edits with webpages that the user is currently viewing. Porta takes a complementary approach by displaying multi-application scent in the browser and not being tied to any specific IDE.

InterTwine [13] connects a web browser and an image editing application. One example form of information scent it provides is to augment Google search result pages for image editing queries with overlays of screenshots and metadata about how the user edited their own images while reading those pages in the past. It requires the GIMP image editor to be modified to add instrumentation code. In contrast, Porta works across arbitrary command-line and GUI applications without needing to modify their code, but its user activity tracing is not as deep as what InterTwine provides for GIMP.

Improving Web-Based Tutorials

The goal of Porta is to help creators improve tutorials in response to how users interact with their content. Porta takes an automated approach by tracking activity across applications. In contrast, systems such as LemonAid [10] and TaggedComments [8] opt for a crowd-based approach by letting users directly annotate parts of documentation and tutorial webpages that appear unclear to them. In the future, we can add support for annotations to complement Porta's automated tracking.

FollowUs [22] implements a comprehensive solution by embedding an instrumented version of a web-based image editing app directly into tutorial webpages. This setup lets users follow tutorials and post their own variant demonstrations directly on the webpage for future users to follow. Porta is motivated by similar goals but does not require such specialized instrumentation. It works on existing unmodified tutorial webpages and requires only installing a macOS tracer app that monitors a variety of other apps without modifying them.

Operating-System-Wide Activity Tracing

Finally, Porta's approach of tracing activity at the operating-system level (rather than within specially-instrumented applications) was inspired by related projects in OS-level tracing.

The closest prior work that takes this approach is Torta [26], which traces operating-system-wide activity to create mixed-media text+video tutorials. That system in turn derives from its predecessor Burrito [17], which performs similar OS-wide tracing to create a personal log of computational activities for researchers to reflect upon. Porta adopts a similar philosophy of OS-wide tracing, but it traces classes of events that prior systems did not, including web browser navigation actions, commands on remote servers accessed via ssh, and invocations of compilers, interpreters, build tools from the command line and within IDEs. Furthermore, Porta's goals are complementary to Torta's: Rather than being used to create new tutorials, Porta is used to profile and help improve *existing tutorials*. Thus, Porta implements a novel web browsing tracer and sidebar overlay for displaying multi-application information scent, which are not present in these prior systems.

PORTA DESIGN AND IMPLEMENTATION

The goal of Porta is to give tutorial creators an efficient way to see how learners actually progress through their instructional materials and where they might have struggled. It is meant to be activated during a user testing session where the participant's task is to follow the steps in a given tutorial.

Porta automatically records relevant actions with no user intervention. At the end of the session, Porta produces a visualization summarizing the participant's actions during each part of the tutorial. In this way, *Porta facilitates user testing of tutorials and other technical documentation* rather than traditional user testing of software artifacts. It consists of three components: 1) an operating-system-wide application usage profiler, 2) a web browsing tracker, and 3) a profiling visualization that augments the original tutorial webpage.

OS-Wide Application Usage Profiler

Porta's OS-wide usage profiler allows it to transparently monitor user activity across multiple applications. Our prototype is implemented for macOS using AppleScript, Python, Bash, and DTrace [9] scripts; but the concept is OS-independent.

When user-testing a tutorial, the participant first launches this profiler in the background before starting to work through the steps on the tutorial webpage. It continually monitors the following data and records it to a JSON-formatted log. All monitored events are timestamped, so in aggregate they form a unified cross-application usage profile.

- **Screencast video**: Apple's built-in Quicktime app is used to record a video of the participant's entire screen, system audio, and spoken audio via the built-in microphone. This is important for showing the participant's actions within GUI-based applications as they follow a tutorial.

- **Clipboard**: When the participant copies text to the clipboard, its contents are logged. This can show what they copied-pasted from tutorials into other apps such as IDEs.

- **Shell command invocations**: Porta installs a wrapper script that logs the timestamps and arguments of all shell commands invoked within terminals. It also logs the current directory and environment variables used for running each command. This logger works for Bash (default on macOS) and zsh, but can easily be extended to other shells.

- **Compiler/interpreter toolchain invocations**: In addition to logging all shell commands, Porta performs deeper tracking when the participant invokes compilers, interpreters, and other build tools (e.g., `make`, `webpack`) as they are following a tutorial. This is important for pinpointing which parts of a tutorial caused the participant to make code errors and tracking certain actions within IDEs.

  Specifically, Porta records the command-line arguments of each tool invocation. It also tracks all the files read by the processes and its forked subprocesses, which are usually the relevant source code files that are compiled or executed. Finally, it records the invocation's textual output (on both stdout and stderr), as well as the error return code, which indicates either a successful or erroneous execution.

  To accomplish this tracking in an application-agnostic way, when Porta is first activated it automatically generates wrapper scripts for toolchain executables such as `gcc`, `make`, `node`, `javac`, and `python` (in a user-customizable list). It adds the directory of those wrappers to the start of the `$PATH` environment variable so that they can be invoked in an identical manner as the original apps.

  When a wrapper script is invoked, it forks a child process to run the original executable with identical command-line arguments. It connects stdin, stdout, stderr streams so that it behaves identically to the original, and also logs the child's stdout/stderr outputs to a file. It then launches DTrace [9] to record `fopen` system calls issued by that child and any of its children. These system calls indicate which files the toolchain invocation is taking as inputs (e.g., Makefiles, source code files). Porta saves a copy of those files in its log directory on each invocation. It ignores binary files (e.g., system libraries) by filtering using MIME types [5].

  DTrace is necessary here since it is impossible to tell what files are accessed by a command solely by inspecting its command-line arguments. For instance, running the command `python foo.py` reads in not only `foo.py` but also all files that are dynamically imported by its code.

  Another major benefit of this wrapper- and DTrace-based approach is that it works regardless of whether these tools are invoked from the terminal or within an IDE. For instance, when the user presses the "Compile" or "Run" button in an IDE, that will invoke the operating system's compiler/interpreter executables, which will call the wrapper versions since those appear first in the user's `$PATH`.

- **Remote activity tracing via `ssh` invocations**: Many kinds of technical tutorials involving system administration, web development, and cloud-based execution will require users to run commands on remote servers. To support remote tracking, Porta replaces the built-in `ssh` executable with a wrapper that injects a shell script into the remote machine when the participant first logs into that machine.

  This injected script lets Porta monitor command invocations on the remote machine in an identical manner as on the participant's local machine. It supports logging into both macOS and Linux servers. Porta uses a Linux port of DTrace to achieve the same kind of toolchain invocation tracing as it does for macOS [14]. In the future, we can also port the tracer to Windows using its Process Monitor [29].

  The ssh script also injects a unique session ID into each login session. This allows Porta to properly group remote toolchain invocations into sessions in cases when the participant either logs into multiple servers or to the same server using different terminal shells. When the user logs out at the end of each session, the ssh script copies the log file from the server back to the participant's local machine.

Note that the related Torta system [26] contains similar screencast video and shell command recorders, but all other components described in this paper are not in Torta. Also, the goal of our Porta system differs from those of prior application profilers: Porta's novelty lies in combining this data with web browser tracking (next section) to create *tutorial profiles*.

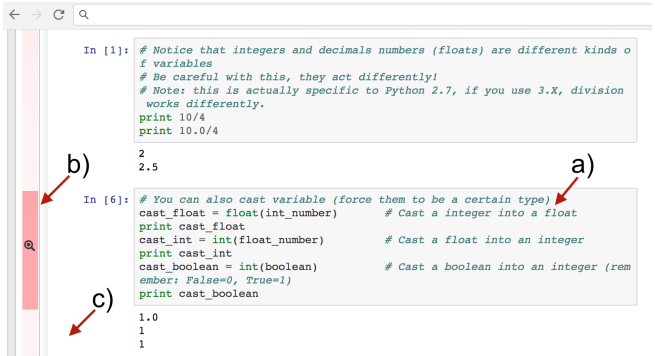

Figure 2. Porta uses mouse location as a proxy for where the user's attention is focused. a) If the user hovers over anywhere in this code block element, Porta will record it as being in focus and b) render it as a red hotspot in the sidebar heatmap. c) If the user hovers over an element (e.g., background) that is larger than the viewport, that event is ignored.

Web Browsing Activity Tracker

Porta also tracks detailed user activity within web browsers using a Google Chrome extension. The participant activates this extension when they are about to start following a tutorial presented on a given webpage. It tracks a timestamped log of the following browser-related activities:

- **Hover-focused webpage element**: The web tracker continually records the mouse position in terms of the most precise CSS path of the DOM element that the participant's mouse is currently hovering over (Figure 2). This provides a rough indicator of what their attention is focused on at each moment. Ideally we would gather data from an eye-tracker to determine the participant's true visual focus, but mouse hover is an approximation that is commonly used in commercial web analytics tools such as FullStory [3], Hotjar [4], and Mouseflow [6]. (Our profile visualizations are designed to account for this level of imprecision.)

  This tracker ignores mouse events over elements that are larger than the browser's viewport (Figure 2c). These positions likely indicate that the mouse is hovering over a webpage background element, which is a weaker indicator of focus, so the tracker conservatively ignores them. It also does not log mouse locations over non-browser windows.

  On the other extreme, when the mouse is hovering over an element that is too small (smaller than 10×10 pixels), the tracker traverses upward in the DOM to the first parent element larger than this minimum size threshold. This heuristic helps ensure it logs that the mouse is hovering over a non-trivial element such as a block of text or an embedded video rather than, say, a tiny stylistic component in the foreground that is occluding more meaningful content.

  Finally, to ignore noisy log entries due to mouse jitter, the tracker does not log an event until the mouse has hovered over a particular element for at least 0.5 seconds.

  Why not simply record raw x-y mouse positions? Because the goal of this tracker is to determine what meaningful page component the user is focusing on at each moment. For example, in Figure 2a, it does not matter whether the user's mouse is over the left or right half of the code block element; we want to record that they are likely focused on that element at the moment. Another benefit of recording DOM elements rather than raw x-y positions is that the former is robust to browser window resizing, which can cause elements to shift to different x-y positions.

  In the end, short of directly asking the user what they are focusing on at each moment in the tutorial, all approximations are imperfect. Even eyetrackers produce noisy data as eye gazes wander and jitter [1]. We wanted to design a simple tracker that works in regular browsers without special equipment, so we adopted this mouse-based approach.

- **Scroll position and viewport size**: The tracker records the current scroll position of the participant in each tab, along with the browser's viewport size. This provides a coarser indicator of where the participant may be currently reading. We assume that if they are reading a webpage, they must be looking at content that is within view of the range defined by the current scroll position and viewport size. (However, we can never know for sure whether the participant is actually reading the tutorial at any given moment.)

- **State of embedded video players**: Technical tutorials sometimes embed short videos alongside their textual content, perhaps to play a mini-lecture, to demonstrate GUI operations, or to show a screencast of code being written and executed. Porta records all events on video player components embedded within webpages, such as play, pause, stop, and seek events, along with their seek positions. This tracer uses the browser's built-in HTML5/JavaScript video API, which works with all modern non-Flash videos including, most commonly, embedded YouTube videos.

- **Opening webpages**: The tracker records the timestamp and URL of every tab and browser window opened by the participant. This is important because they might open new tabs to search for topics in the tutorial that are hard for them to understand, so Porta should track when they do so.

- **Opening Chrome developer tools**: It also records the tabs in which the participant has opened the browser's developer tools pane, which contains a JavaScript console, HTML/CSS inspector, network inspector, and JavaScript debugger. This action signals that they may be trying to follow some kind of web development tutorial and are currently debugging their web-related code in that tab.

- **JavaScript errors**: Finally, the tracker logs all JavaScript errors on webpages where the participant has opened the developer tools to presumably debug their web-related code while following a tutorial. In addition, errors are always logged for pages on the localhost domain, even without opening developer tools, since those are likely pages that the participant is editing and debugging locally while following a web development tutorial. This within-browser logging is similar to Porta logging the error messages produced by compiler/interpreter toolchain invocations.

Although profiling user activity within webpages is not a new idea (commercial analytics tools do some form of this [3, 4, 6]), the main novelty of Porta lies in combining web tracking with the application usage profiler from the prior section to create *tutorial profiling visualizations*.

OS-Wide Application Usage Profiler
Screencast video (fullscreen audio/video recording of test session)
Clipboard text (for tracking copy-paste actions)
Shell commands (all commands run in Bash/zsh in any terminal)
Toolchain invocations (run from shell or within an IDE)
Remote ssh invocations (shell/toolchain commands on remote servers)
Web Browsing Activity Tracker
Hover-focused webpage element (use mouse as proxy for user focus)
Scroll position and viewport size
Embedded video player state (all HTML5 players including YouTube)
Opening webpages
Opening Chrome developer tools
JavaScript errors

Table 1. Summary of trace data that is automatically collected by Porta's OS-wide and within-browser trackers. All events are timestamped.

Tutorial Profiling Visualizations

Table 1 summarizes the data that Porta automatically collects on the participant's computer as they follow a tutorial during a testing session. Porta combines all this data into a tutorial profiling visualization, which can be used in two main ways:

- The test facilitator shows it to the participant in a post-study debriefing so that the participant can see where they struggled and better reflect on why they took those actions.
- Porta can aggregate the trace data from a group of test users and present it to the tutorial's creator so that they can see where people collectively struggled.

In both use cases, the ultimate goal is to provide feedback to the tutorial's creator so that they can improve its contents.

Heatmap visualizations: Figure 3 shows that Porta displays tutorial profiles as positional and temporal heatmaps alongside the left and bottom of the tutorial webpage, respectively. The right half of Figure 1 shows this UI on a real webpage.

We implemented this visualization as a web application that embeds the original tutorial webpage in an iframe. We took this iframe-based approach so that we do not need to modify the tutorial webpages. In earlier iterations of Porta, we injected DOM elements and JavaScript events as an overlay atop webpages, but that was not robust; our elements sometimes altered the layout of those pages or came into conflict with frontend frameworks or libraries. Overlays also occluded certain page elements. Using our iframe approach, the profiled webpages appear exactly as how the test participant and tutorial creator originally saw them.

The *positional heatmap* shows where on the tutorial webpage the participant was looking at the most, and what else they were doing on their computer while looking at each part. Just like how a source code profiler [16, 31] shows *hotspots* of lines of code where the computer spent the most time executing, this heatmap visualization shows hotspots of where participants spent the most time while following a tutorial.

This heatmap shows the approximate amounts of time that the participant spent on each vertical portion of the webpage throughout the session. Figure 4 shows how it is the time-weighted sum of two components: 1) a precise component based on mouse hover locations over specific DOM elements, and 2) a coarse component based on browser scroll position

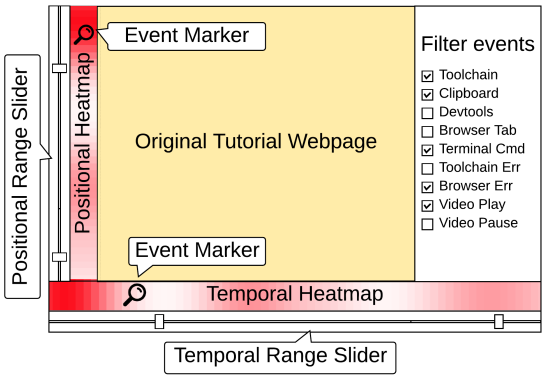

Figure 3. Porta uses the trace data collected from test sessions to augment the original tutorial webpage with heatmap visualizations showing positional focus and temporal density of events. Specific occurrences of events show up as clickable markers on both heatmaps.

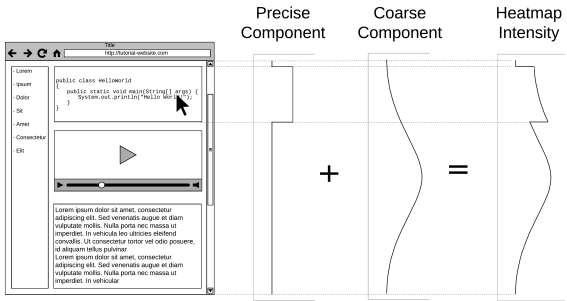

Figure 4. The positional heatmap's intensity is a time-weighted sum of precise (mouse hovering over a specific DOM element) and coarse components (Gaussian centered at the middle of the viewport).

and viewport size. For example, in Figure 4, the mouse is hovering over an example code block, so the precise component covers the entire height of that DOM element. However, we cannot be certain that the participant is actually looking at that element; they might have left the mouse there while reading other content on the page. To account for this inherent uncertainty, we add a coarse component consisting of a Gaussian distribution at the middle of the viewport. We chose parameters to cover the viewport within 2 standard deviations (95% of the Gaussian's area). The assumption here is that the user is more likely looking at the center rather than the edges.

If the participant now moves their mouse away from the browser to another application window such as their IDE, we can no longer use mouse location as a proxy for their current focus. Thus, Porta instead relies only on the coarse (Gaussian) component to approximate their focus location. If the browser window is occluded, that may decrease the chances that they are looking at certain parts of the webpage, but we have not yet accounted for this level of detail in our prototype.

To produce the positional heatmap for the entire session, Porta time-weighs the computed values for each moment and maps them to a monochromatic color scale to display a 1-D vertical heatmap along the left side of the tutorial webpage. Since this is located in a separate iframe, Porta synchronizes vertical scroll events and viewport size changes between the tutorial webpage iframe and the heatmap's iframe so both are always in sync regardless of page scrolling or resizing. We

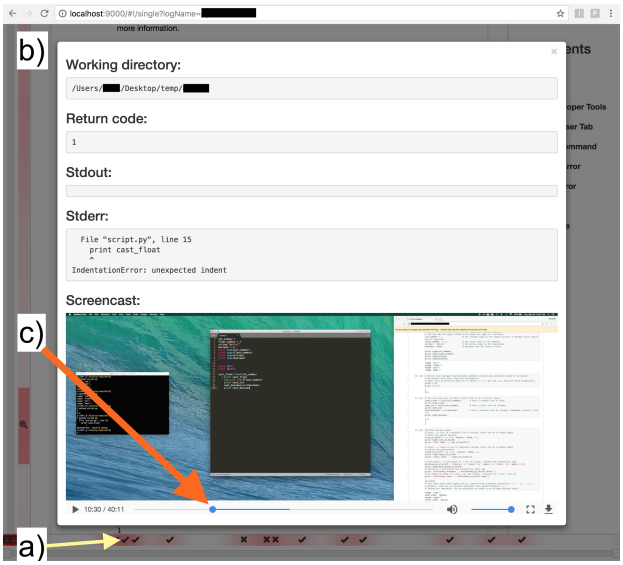

Figure 5. Event markers and pop-ups: a) When the user clicks on an event marker on a heatmap, Porta will b) pop up a dialog showing details about that event. c) This dialog also includes a fullscreen video of the test session that starts playing 20 seconds before that event occurred.

chose a 1-D vertical heatmap since tutorials are usually formatted vertically. This also matches the interface of code profilers, which display vertically down an IDE's or text editor's gutter to show which lines of code were executed the most.

Similarly, a *temporal heatmap* along the bottom of the tutorial webpage shows the density of events logged from Table 1 throughout the time duration of the testing session. This lets viewers get a sense for the temporal ordering of events in the testing session and filter by time ranges (described later).

Event markers and pop-ups: Heatmaps show an overview of the session's activity, but it is also important to see exactly which actions the participant performed while they were focusing on each part of the tutorial. To surface this data, Porta displays an *event marker* for each type of event in Table 1 at the approximate webpage location where the user was looking when they performed that action. When mouse hover data is available, this marker is placed at the center of the focused DOM element; when it is not available, the marker is placed at the center of the viewport's vertical position. An identical copy of that marker appears on the temporal heatmap as well.

When the user clicks on an event marker on either the positional or temporal heatmap (Figure 5a), a pop-up dialog appears to show the details of that event (Figure 5b). This dialog contains an embedded screencast video recording of the entire session with its seek position set to 20 seconds prior to that event's occurrence (Figure 5c). This way, viewers can see the context leading up to the selected event.

It also displays more detailed contextual data depending on event type (Figure 5b): Clipboard events show the textual contents that were copied at the time of occurrence. Shell commands, toolchain invocations, and remote ssh invocations show all of the logged data, including command-line arguments, textual outputs, error messages, return codes, and the contents of the affected files at the time that command was run. Likewise for browser actions: If a new webpage tab or window was opened, it shows that page's URL; if any JavaScript errors arose, those are also shown.

If an event occurred when an embedded video on the tutorial webpage was playing (or was paused at a non-starting position), then its pop-up dialog also displays a video player that loads that video at the same position. This level of detail is important for video-centric tutorials: Without it, viewers can see only that the event occurred when the mouse was hovering over an embedded video element on the tutorial webpage, but not where the participant was watching *within the video* when they performed an action that logged that event.

Filtering to reduce visual overload: By default all events are shown on both the positional and temporal heatmaps. Although their positions are slightly jittered to prevent direct overlap, if there are too many events, it may still be hard to select individual markers. To mitigate this problem, Porta allows viewers to filter by event type using facets (checkboxes), which will show only selected kinds of events on heatmaps.

Viewers can also filter by time. By dragging a double-ended range slider on the temporal heatmap, they can hone in on a time range. This will show markers for only events that occurred within that range in *both* the temporal and positional heatmaps. It will also re-render the positional heatmap to show the relative amounts of time that the participant viewed portions of the webpage during that selected time range.

Viewers can also filter by position using a similar double-ended range slider on the positional heatmap. This will again filter events on both heatmaps to show only those within the selected range. Another benefit of positional filtering is being able to zoom in on a detailed heatmap about a specific portion of the webpage. One limitation of showing a single global heatmap that spans the entire webpage is that certain regions may be too close in value and thus appear almost as the same color. When the user selects a positional range, the heatmap will be computed only for vertical positions within that range, which will amplify those subtle value differences.

Aggregating multiple sessions: Porta can aggregate the data collected from multiple test sessions in a simple way. It overlays all of the trace data on the positional heatmap so that it visualizes the relative amounts of time spent by all participants on each portion of the tutorial webpage. This is akin to a source code profiler showing aggregated results from multiple independent executions. Likewise, event markers from all participants are shown on the heatmap. To avoid further visual overload, Porta adds an additional facet so that the viewer can filter by participant ID as well as by event type.

We chose not to display a temporal heatmap when aggregating multiple sessions since each participant likely takes differing amounts of time to work through the tutorial and perform their actions in different orders. An alternative design is to display separate temporal heatmaps for each participant, but that may lead to even more visual overload. If a viewer wants to inspect an individual participant's session in detail, they can view it in isolation in its own browser window.

DISCUSSION: SYSTEM SCOPE AND LIMITATIONS

Porta is best suited for profiling multi-application tutorials commonly found in domains such as software engineering, web development, system administration, and data science. It is less well-suited for detailed profiling of single-application tutorials such as those for MS Word, Excel, or Photoshop. Since Porta does not perform tracking within GUI apps, the most it can do for those tutorials is display a general heatmap and screencast videos. One way to overcome this limitation is to build plug-ins that track these applications in detail.

We envision two main classes of use cases: testing newly-written tutorials and updating more mature tutorials, which can suffer from *bit rot* over time as target users' computer configurations change. Creators may be motivated to use Porta to make sure that older tutorials keep working properly.

There is inherent imprecision to Porta's heatmaps since it is impossible to tell where in the tutorial the participant is truly focusing on, short of continually asking them. The Gaussian distribution is a crude model for approximating uncertainty in positional focus. Time estimates are also imprecise because Porta cannot easily tell whether the user is on task, distracted, or taking a break. On a related note, is spending more time on a certain part of the tutorial good or bad? Maybe it is good since that part is more engaging, or maybe it is bad if there are lots of error-related events at those parts. Thus, profile visualizations should be used ***as a substrate to facilitate discussion and reflection, not as a source of precise ground truth***.

Porta requires installing and running OS-wide monitoring software that may lead to privacy concerns. When used in an in-person user test, it can come pre-installed on the lab computer where session state gets erased after each trial. When used in online experiments, participants must both be technically adept enough to install it and put their trust in it. To respect privacy, Porta starts recording in a new empty Chrome browser profile only when the user explicitly activates it, stores data only on the computer it is installed on, and never transmits data remotely without consent. The user can inspect all raw data that Porta collects and view the profile visualizations, which is exactly what the tutorial creator sees. They can manually delete portions of the logs or video files that they do not want to share, or choose not to share at all. These complexities mean that Porta is not well-suited for longer-term always-on deployment in a field study. Rather, we envision it being selectively activated only during user testing sessions, whether in-person or remote. In the end, participants must determine if the potential benefits (e.g., altruism, paid compensation) outweigh the privacy costs of usage.

Since Porta's positional heatmaps are vertically aligned, it is not designed for horizontally-scrolling tutorials or those that dynamically render content without scrolling (e.g., using an image carousel or flipbook metaphor). In our experience, though, these formats are rarely seen in software tutorials.

Finally, Porta displays heatmaps and event markers for only one tutorial webpage at a time. If a tutorial spans multiple webpages, then viewers must view each profile webpage separately. Inter-page linking is one direction for future work.

EVALUATION: USER STUDIES

To assess Porta's potential efficacy, we had 12 students activate it while following 3 software tutorials. We then showed the resulting Porta outputs to the instructors who created those tutorials. We wanted to investigate two main questions:

- Does Porta help students better reflect on the difficulties they faced while following tutorials?
- Does Porta provide useful feedback to instructors about how to improve their own tutorials in the future?

Materials: For this study, we used three web-based tutorials created by instructors at our university for classes they teach: 1) *Python*: A primer on basic Python types, control flow, and functions; created for a data science course (~800 words). 2) *Git*: Intro. to the Git version control system and GitHub [21]; created as part of a MOOC on introducing scientists to command-line development tools (~3,000 words). 3) *Web Design*: Intro. to HTML, CSS, and JavaScript with jQuery; created for an HCI course (~500 words).

Each tutorial was formatted as step-by-step instructions on a single vertically-scrolling webpage with mini-exercises for students to check their understanding. The Web Design tutorial also featured embedded screenshots and mini-videos.

Procedure for Student User Study: We recruited 12 computer science undergraduate students (9 women) from our university each for a 1-hour user study; each was paid $10. To find novices, we limited recruitment to those who had little to no experience with the subject of the tutorial they saw. Each participant came to our lab to work through one tutorial on a macOS machine with both Porta and the necessary software (e.g., text editors, terminal app, Python, Git) installed:

1. We activated Porta and gave the participant up to 40 minutes to work through the tutorial in any way they wished.
2. After stopping Porta, we asked the participant to reflect on any difficulties they faced while following the tutorial and to provide suggestions for improving the tutorial. Note that this debriefing occurs *before* they ever see Porta's output.
3. Finally, we showed the participant the output of Porta and let them freely explore the profile visualization interface. Throughout this process, we asked them to further reflect on any suggestions they have for improving the tutorial.

Procedure for Instructor User Study: After completing the student user study, each of the 3 tutorials now had profile information collected from 4 students trying to follow them. For this study, we had the instructors who created each tutorial come to our lab for one hour and inspect Porta's output:

1. We began by showing the instructor their own tutorial and having them reflect on if they wished to make any changes to it. Note that this occurs *before* they ever see Porta.
2. We then showed the instructor Porta outputs from each of the 4 student studies, as well as the aggregate visualization of all 4 sessions together. We let them freely explore the interface. We asked them to think aloud and again reflect on whether they wished to make any changes to their tutorial.

Study Limitations: Our findings came from self-reported anecdotes from first-time users. We have no evidence of longitudinal effects such as whether the instructors actually made the suggested improvements to their tutorials or whether future students ended up benefiting from those improvements. We also opted for a within-subjects study design so that we could directly compare the nature of each participant's feedback before and after they saw Porta's output. There may be some ordering effects, but it is infeasible to flip the order of exposure: i.e., if we first show someone Porta's output, then they cannot "un-see" it later in the session. Also, due to time constraints in study sessions, we opted not to have each instructor first view a baseline condition where they would have needed to watch 4×40-minute raw screencast videos from their respective student sessions. Note that instructors can still view selected excerpts of raw student videos within Porta by clicking on relevant events in the UI (Figure 5c).

Findings from Student User Study

Table 2 summarizes Porta's recordings for the 12 participants in the student user study (P1–P12). Everyone completed their tutorial within the 40 minutes they were given. Since we did not formally assess students' understanding of the subjects, it is possible that they made mistakes while performing the given actions or harbored some misconceptions; however, we feel that this is a realistic simulation since students would not be supervised when following these tutorials on their own.

Table 2 also shows the occurrences of events that Porta recorded during each session. Taken together, these three tutorials elicited all event types: The Python tutorial involved running shell commands, the Git tutorial involved ssh-based commands to use Git on a remote server, and the Web tutorial involved Chrome developer tool interactions. Although we did not formally measure application run-time speeds, participants did not report any performance-related problems.

Feedback comparison: Table 3 contrasts the qualitative feedback that participants provided before and after seeing Porta's output. Before seeing Porta's output, they provided either vague or non-existent feedback. We gave each one the opportunity to look through their tutorial again, and 8 out of 12 participants felt like it was good enough in its current state. For instance, P1 said that the Python tutorial was "easy to follow" and P5 said the Git tutorial "seemed straightforward." When they did offer critiques, their descriptions were high-level: e.g., "language could be more novice friendly" (P8).

In contrast, once participants started exploring Porta's output, they were able to give much more specific and targeted feedback. Everybody had at least one concrete suggestion for improvement, even those who minutes earlier had just said that the tutorial looked fine as-is. The upper right of Table 3 shows one example from each participant. Aside from being specific, each suggestion was made while referencing a specific location in the tutorial, so they were precisely targeted.

For example, both P3 and P4 originally said the tutorial looked fine, but as they explored Porta's output they zoomed in on occurrences of errors while running Python commands. They saw that there were error messages related to them using

Participant	Time	# application events			# web browser events		
		Local (errors)	Remote	Clip	Pages	Devtools	Errors
P1 Py	38	50 (10)	0	0	0	0	0
P2 Py	26	35 (1)	0	23	0	0	0
P3 Py	26	36 (5)	0	12	0	0	0
P4 Py	29	30 (3)	0	18	0	0	0
P5 Git	24	0	57	0	0	0	0
P6 Git	28	0	73	0	0	0	0
P7 Git	21	0	52	0	0	0	0
P8 Git	21	0	49	33	0	0	0
P9 Web	28	0	0	9	2	0	2
P10 Web	17	0	0	0	2	0	5
P11 Web	21	0	0	8	4	1	3
P12 Web	37	0	0	0	10	1	2

Table 2. Summary of Porta events recorded during the 12 student user study sessions for Python, Git, and Web Design tutorials. Session time is in minutes. "Local" events include both shell commands and compiler/interpreter toolchain (e.g., Python) invocations. "Clip" is clipboard copy-and-paste. "Pages" is opening additional webpages in new tabs.

"true" and "false" for booleans instead of the properly capitalized versions ("True" and "False") that Python requires. They suggested for the tutorial creator to add a clarifying note there to help students who were used to bools in other languages.

In theory, participants could glean this same information from watching a raw screencast video recording of their sessions, but it would likely be harder to pinpoint occurrences of key events in a 40-minute-long video. Porta's visualizations allowed participants to quickly zoom in on key events such as command invocations and toolchain errors so that they watched only the video segments centered at those events. Thus, it provided a convenient event- and time-based index into the underlying raw screencast videos that it recorded.

Findings from Instructor User Study

The ultimate goal of Porta is to give useful feedback to tutorial creators. To assess how well it achieves this goal, we showed the instructor who created each tutorial the Porta outputs from all 4 of its student user study sessions. (To evaluate Porta in isolation, we did *not* show instructors the actual feedback that students provided; they saw only Porta's outputs.)

To get baseline impressions, before introducing Porta we asked each instructor to look over their tutorial and let us know if they wanted to make any specific changes to it. As the lower left of Table 3 shows, they provided only vague ideas such as "split some of the sections up." All three felt their tutorial was in good shape since it had been used by many past students: The Python tutorial had been used in two iterations of a 400-student data science course; the Web Design one was used in four iterations of a 300-student HCI course; and the Git tutorial was featured in a 10,000-student MOOC. Thus, these instructors had already fixed many issues from having these tutorials so heavily used over the past few years.

Expert blind spot effect: While exploring Porta's visualizations, all three instructors noticed unexpected student behavior that surprised them, even despite having taught multiple times using these instructional materials. This could be an instance of the expert blind spot [27] whereby experts have trouble relating to what novices know and do not know since, as experts, they are too familiar with their own subject matter.

Students		Feedback on tutorial (before seeing Porta's output)	Feedback on tutorial given while using Porta
P1	Py	"easy to follow"	"mention the variable in a 'for' loop is a value and not an index"
P2	Py	"tutorial is nice"	"colons are used to start code blocks in Python"
P3	Py	"I didn't know where to start writing the code"	"boolean values start with capital letters in Python"
P4	Py	"a summary of the tutorial would be good."	"boolean values should be capitalized in Python"
P5	Git	"seemed straightforward"	"talk about how to exit the 'less' program when showing 'git status'"
P6	Git	"need more Windows-specific advice"	"explain what the 'cat' command does here"
P7	Git	"some parts did not clearly explain other tools used in the tutorial"	"explain the git staging step better"
P8	Git	"the language could be more novice friendly"	"use a different formatting for text and commands"
P9	Web	"some screenshots were too small to read"	"show large CSS changes that can be seen in screenshots"
P10	Web	"more details about Bootstrap"	"add more comments in the starter code about what's going on"
P11	Web	"more explanation about what was supposed to happen"	"no explanation of where event object for click handler comes from"
P12	Web	"more explanation about the lines they added"	"make the file in which code is to be written more clean"

Instructors	What they want to change in their tutorial (before using Porta)	What they want to change, mentioned while using Porta
Python tutorial creator	"update this for Python 3" "in the past, students had trouble with looping in this tutorial. look into how to do that part better."	"talk about needing to start code blocks with colons & indentation" "split text into inline comments which the students actually read" "didn't realize escape sequences threw people off; add more explanation about that" "introduce basic Python syntax before talking about types"
Git tutorial creator	"tutorial is probably too long" "split some of the sections up"	"use HEAD~1 instead of HEAD~ because it's more clear that you're going 1 commit back" "don't use the intentional "fil2" typo; none of the students got it" "make it more clear that students should use their own user names and email addresses in examples"
Web tutorial creator	"link to more external content" "reformat some of the steps for better flow"	"don't use HTML placeholders in code; students copy them literally" "show that capitalization matters when linking external files" "make more obvious CSS changes so students can actually see something happening" "reverse the order of changes in the CSS of steps 7, 8, and 9"

Table 3. Examples of feedback given by the 12 students (top) and 3 instructors (bottom) on each tutorial, both before seeing Porta (left) and while using Porta (right). Feedback while using Porta was usually more concrete, specific, and precisely targeted to one particular location within the tutorial.

All three explored Porta's positional heatmaps to see where students spent relatively more time. Each clicked on an average of 31 event markers and spent 33% of their total session time viewing event details; for reference, each of the 12 students clicked an average of 6.8 event markers and spent 17% of time viewing details in their own sessions. While we do not have rigorous data on mouse tracking accuracy, when participants were reviewing their own heatmaps, there were no times at which they reported being surprised by false positives.

Table 3 shows that all three gained actionable insights. For instance, the Python tutorial's creator did not realize that syntax to demarcate code blocks with colon and whitespace was a major hurdle for novices; he realized this only when he saw several students struggling with indentation errors and repeatedly having those error events show up in Porta's output. The Git tutorial's creator had mastered Unix command-line tools, so he did not anticipate that students would have such a hard time using basic commands such as 'less' and 'cat'.

Ideas for improving tutorials: Instructors were also able to use Porta to come up with concrete ideas for improving their tutorials. The lower right of Table 3 summarizes their ideas.

For instance, the Python tutorial's creator saw from heatmap visualizations that most students did not even read through major parts of his tutorial. He realized that reformatting those parts as inline comments in code examples might work better. Also, from observing the temporal order of events, he came to the conclusion that he should have introduced code blocks and whitespace significance in the tutorial first before introducing Python types. The Git tutorial's creator was surprised that adding in an intentional typo for "fil2" tripped up all the students who encountered it. He expected students to run the command verbatim with the typo, but everyone actually used the correct spelling of "file2" and therefore got confused by the next section that explained the consequences of the intentional mistake.The Web Design tutorial's creator saw that he should have made CSS style changes more visually salient.

In theory, instructors could have gleaned these insights via direct observation or by watching videos of test sessions. However, needing to directly observe users limits scale, whereas Porta could be used to run user tests remotely and be administered by third parties. It would also likely take them much longer to watch the raw videos, and they would not get the benefits of Porta's heatmaps or event markers to hone in on clusters of related user activities. Finally, Porta provides a compact summary of test sessions that can easily be shared with other people such as co-instructors or future students.

CONCLUSION

This paper addresses the challenges of providing effective, fine-grained feedback on the contents of software tutorials. To do so, we created Porta, a system that automatically builds *tutorial profiles* by tracking user activity within a tutorial webpage and across multiple applications on their computer. Porta surfaces these profiles as interactive visualizations that show hotspots of user focus alongside details of logged application events and embedded segments of recorded screencast videos. In sum, Porta opens up possibilities for systematic user testing of technical documentation at scale by providing fine-grained data to both test participants and tutorial creators.

ACKNOWLEDGMENTS

Thanks to Kandarp Khandwala, Sean Kross, Xiong Zhang, and the UCSD Design Lab for their feedback.

REFERENCES

1. 2011. Accuracy and precision test method for remote eye trackers. https://www.tobiipro.com/siteassets/tobii-pro/accuracy-and-precision-tests/tobii-accuracy-and-precisiontest-method-\version-2-1-1.pdf. (2011).
2. 2018. Flask (A Python Microframework). http://flask.pocoo.org/. (2018).
3. 2018. FullStory: Pixel-Perfect Session Replay. https://www.fullstory.com/. (2018).
4. 2018. Hotjar: All-in-one Analytics & Feedback. https://www.hotjar.com/. (2018).
5. 2018. MDN web docs: MIME types. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. (2018).
6. 2018. Mouseflow - Session Replay, Heatmaps, Funnels, Forms & User Feedback. https://mouseflow.com/. (2018).
7. Jason Alexander, Andy Cockburn, Stephen Fitchett, Carl Gutwin, and Saul Greenberg. 2009. Revisiting Read Wear: Analysis, Design, and Evaluation of a Footprints Scrollbar. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1665–1674. DOI: http://dx.doi.org/10.1145/1518701.1518957
8. Andrea Bunt, Patrick Dubois, Ben Lafreniere, Michael A. Terry, and David T. Cormack. 2014. TaggedComments: Promoting and Integrating User Comments in Online Application Tutorials. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 4037–4046. DOI: http://dx.doi.org/10.1145/2556288.2557118
9. Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Leventhal. 2004. Dynamic Instrumentation of Production Systems. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '04)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=1247415.1247417
10. Parmit K. Chilana, Andrew J. Ko, and Jacob O. Wobbrock. 2012. LemonAid: Selection-based Crowdsourced Contextual Help for Web Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1549–1558. DOI: http://dx.doi.org/10.1145/2207676.2208620
11. Biplab Deka, Zifeng Huang, Chad Franzen, Jeffrey Nichols, Yang Li, and Ranjitha Kumar. 2017. ZIPT: Zero-Integration Performance Testing of Mobile App Designs. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 727–736. DOI: http://dx.doi.org/10.1145/3126594.3126647
12. Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction Mining Mobile Apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 767–776. DOI: http://dx.doi.org/10.1145/2984511.2984581
13. Adam Fourney, Ben Lafreniere, Parmit Chilana, and Michael Terry. 2014. InterTwine: Creating Interapplication Information Scent to Support Coordinated Use of Software. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 429–438. DOI: http://dx.doi.org/10.1145/2642918.2647420
14. Paul D. Fox. 2018. Linux port of DTrace. https://github.com/dtrace4linux/linux. (2018).
15. Max Goldman and Robert C. Miller. 2009. Codetrail: Connecting Source Code and Web Resources. *J. Vis. Lang. Comput.* 20, 4 (Aug. 2009), 223–235. DOI: http://dx.doi.org/10.1016/j.jvlc.2009.04.003
16. Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. 1982. Gprof: A Call Graph Execution Profiler. In *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction (SIGPLAN '82)*. ACM, New York, NY, USA, 120–126. DOI: http://dx.doi.org/10.1145/800230.806987
17. Philip J. Guo and Margo Seltzer. 2012. BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure. In *Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance (TaPP'12)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=2342875.2342882
18. Björn Hartmann, Mark Dhillon, and Matthew K. Chan. 2011. HyperSource: Bridging the Gap Between Source and Code-related Web Sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2207–2210. DOI: http://dx.doi.org/10.1145/1978942.1979263
19. William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. 1992. Edit Wear and Read Wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*. ACM, New York, NY, USA, 3–9. DOI: http://dx.doi.org/10.1145/142750.142751
20. Juho Kim, Philip J. Guo, Carrie J. Cai, Shang-Wen (Daniel) Li, Krzysztof Z. Gajos, and Robert C. Miller. 2014. Data-driven Interaction Techniques for Improving Navigation of Educational Videos. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 563–572. DOI: http://dx.doi.org/10.1145/2642918.2647389

21. Sean Kross. 2017. The Unix Workbench. Chapter 6: Git and GitHub. https://seankross.com/the-unix-workbench/git-and-github.html. (2017).
22. Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. 2013. Community Enhanced Tutorials: Improving Tutorials with Multiple Demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1779–1788. DOI: http://dx.doi.org/10.1145/2470654.2466235
23. Tom Lieber, Joel R. Brandt, and Rob C. Miller. 2014. Addressing Misconceptions About Code with Always-on Programming Visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2481–2490. DOI: http://dx.doi.org/10.1145/2556288.2557409
24. Justin Matejka, Tovi Grossman, and George Fitzmaurice. 2013. Patina: Dynamic Heatmaps for Visualizing Application Usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 3227–3236. DOI: http://dx.doi.org/10.1145/2470654.2466442
25. Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. Bifröst: Visualizing and Checking Behavior of Embedded Systems Across Hardware and Software. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 299–310. DOI: http://dx.doi.org/10.1145/3126594.3126658
26. Alok Mysore and Philip J. Guo. 2017. Torta: Generating Mixed-Media GUI and Command-Line App Tutorials Using Operating-System-Wide Activity Tracing. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 703–714. DOI: http://dx.doi.org/10.1145/3126594.3126628
27. Mitchell J Nathan, Kenneth R Koedinger, and Martha W Alibali. 2001. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In *Proceedings of the third international conference on cognitive science*. Beijing: University of Science and Technology of China Press, 644–648.
28. Peter L. T. Pirolli. 2007. *Information Foraging Theory: Adaptive Interaction with Information* (1 ed.). Oxford University Press, Inc., New York, NY, USA.
29. Mark Russinovich. 2018. Microsoft Sysinternals: Process Monitor v3.50. https://docs.microsoft.com/en-us/sysinternals/downloads/procmon. (2018).
30. Victor Savkin. 2016. Angular: Why TypeScript? https://vsavkin.com/writing-angular-2-in-typescript-1fa77c78d8e8. (2016).
31. Amitabh Srivastava and Alan Eustace. 1994. ATOM: A System for Building Customized Program Analysis Tools. In *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation (PLDI '94)*. ACM, New York, NY, USA, 196–205. DOI: http://dx.doi.org/10.1145/178243.178260