UNIVERSITY OF CALIFORNIA, SAN DIEGO

Operating System Wide Activity Tracing for Generating and Profiling Software Tutorials

A dissertation submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Alok Shankar Mysore

Committee in charge:

Professor Philip J. Guo, Chair
Professor Scott R. Klemmer
Professor Leo Porter

2018

The dissertation of Alok Shankar Mysore is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2018

DEDICATION

To my brother, mom, dad and step-dad,

I wouldn't be here without you!

TABLE OF CONTENTS

Signature Page iii

Dedication iv

Table of Contents v

List of Figures vii

List of Tables ix

Acknowledgements x

Abstract xi

Chapter 1 Introduction 1

Chapter 2 Related Work 4
2.1 Operating-System-Wide Activity Tracing 4
2.2 Related work for Torta 5
2.2.1 Mixed-Media Tutorials that Combine Text and Video 5
2.2.2 Generating Tutorials by User Demonstration 6
2.3 Related work for Porta 7
2.3.1 Application-Specific Usage Profiling 7
2.3.2 Multi-Application Information Scent 9
2.3.3 Improving Web-Based Tutorials 9

Chapter 3 Torta: Generating Software Tutorials Using Operating System Wide Activity Tracing 11
3.1 Introduction 11
3.2 Formative Interviews and Design Goals 12
3.3 The Design and Implementation of Torta 15
3.3.1 Tutorial Recorder 15
3.3.2 Tutorial Editor 20
3.3.3 Tutorial Viewer 23
3.4 Exploratory User Studies 25
3.4.1 Tutorial Creator User Study 26
3.4.2 Tutorial Consumer Pilot Study 29

3.5 Discussion of Design Space and Limitations 31
3.6 Acknowledgment 34

Chapter 4 Porta: Profiling Software Tutorials Using Operating System Wide Activity Tracing 35
4.1 Introduction 35
4.2 Porta Design and Implementation 35
4.2.1 OS-Wide Application Usage Profiler 36
4.2.2 Web Browsing Activity Tracker 39
4.2.3 Tutorial Profiling Visualizations 43
4.3 Evaluation: User Studies 50
4.3.1 Findings from Student User Study 52
4.3.2 Findings from Instructor User Study 54
4.4 Discussion: System Scope and Limitations 58
4.5 Acknowledgment 59

Chapter 5 Conclusion 61

Chapter 6 Future work 63

Bibliography 64

LIST OF FIGURES

Figure 3.1: A mixed-media tutorial for web programming that was manually created for an HCI course [Kle17]. Each tutorial is up to 100 steps consisting of PowerPoint slides mixed with code snippets and videos. 12
Figure 3.2: Torta allows macOS users to create mixed-media tutorials by demonstration, and then edit, view, and run those tutorials in a web browser. 15
Figure 3.3: Example structure of a mixed-media tutorial generated by Torta. Each of the three steps represents a foreground GUI window duration. There are three sub-steps within the terminal and IDE windows. 18
Figure 3.4: Zoomed-in screenshots of Torta's tutorial editor showing three steps (i.e., foreground windows): iTerm2, Google Chrome, and Sublime Text. Each step contains: a.) Video player with playback speed adjuster, b.) Text annotation box, c.) Toggle to show/collapse/hide this entire step, d.) Toggle to show/collapse/hide each sub-step (here the hidden shell command sub-steps are crossed-out), e.) Validation script, f.) Filesystem tree (see Figure 3.5). 19
Figure 3.5: Each file-modifying tutorial step displays a filesystem tree of all files affected by running that step. In the editor UI, the user can right-click to show/hide files and to mark for validation. 21
Figure 3.6: Each step and sub-step contains "Validate" and "Run" buttons on the upper right. Here when the user clicks on "Run" for a shell command sub-step, Torta runs that commands in a new terminal window. 24
Figure 3.7: The design space of tools for generating step-by-step records of application actions, ranging from those that automatic run (i.e., scripts) to those meant for users to manually follow their steps (i.e., tutorials). 33

Figure 4.1: Porta provides feedback to tutorial creators about how users navigated through their tutorials and what application errors they encountered. 36

Figure 4.2: 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. 40
Figure 4.3: 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. 45
Figure 4.4: 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). . 46
Figure 4.5: 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. 47
Figure 4.6: Screenshots of all 3 tutorials with green arrows pointing to the improvements identified by the learners and pink arrows pointing to the improvements identified by the creators 60

LIST OF TABLES

Table 3.1: Tutorial creator study results, showing subject IDs, which tool they used first, summary of their tutorial, time in each tool, and the numbers of steps in Torta and GDoc tutorials († did not explicitly denote steps in GDocs). All times reported in minutes, with Torta split into recorder+editor times. 25

Table 4.1: Summary of trace data that is automatically collected by Porta's OS-wide and within-browser trackers. All events are timestamped. 44

Table 4.2: 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. 54

Table 4.3: 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. 55

ACKNOWLEDGEMENTS

I am extremely privileged to have many exceptional people around me who have inspired and supported me.

My advisor and chair, Philip for funding and guiding my research. He has truly been a great mentor and a friend and his work ethic is what I aspire to cultivate.

My committee members, Prof. Scott Klemmer and Prof. Leo Porter for their insight and for their feedback.

My friends, Swathi and Tharun for being constant sources of amusement and my friend and lab-mate, Kandarp for all the sugar highs.

ABSTRACT OF THE DISSERTATION

Operating System Wide Activity Tracing for Generating and Profiling Software Tutorials

by

Alok Shankar Mysore
Master of Science in Computer Science

University of California San Diego, 2018

Professor Philip J. Guo, Chair

Programming tutorials are vital for helping people perform complex software-based tasks in domains such as programming, data science, system administration, and computational research. However, it is tedious to create detailed step-by-step tutorials for tasks that span multiple interrelated Graphical User Interface (GUI) and command-line applications. Once these tutorials are created, it is also 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.

We address these challenges by creating two prototypes:

- **Torta** - an end-to-end system that automatically generates step-by-step GUI and command-line app tutorials by demonstration, provides an editor to trim, organize, and add validation criteria to these tutorials, and provides a web-based viewer that can validate step-level progress and automatically run certain steps.

- **Porta** - a system that automatically tracks how users navigate through a tutorial webpage and what actions they take on their computer. Porta tracks

running shell commands, invoking compilers, and logging into remote servers and surfaces this trace data in the form of profiling visualizations. The visualization augments the tutorial with heatmaps of activity hotspots and markers that expand to show event details, error messages, and embedded screencast videos of user actions.

The core technical insight that underpins both these systems is that Operating-system-wide activity tracing makes it possible to easily generate new tutorials and profile existing tutorials.

An exploratory study on 10 computer science teaching assistants (TAs) and 6 students found that they all preferred the experience and results of using Torta to record and consume tutorials respectively.

A user study of 3 tutorial creators and 12 students who followed their tutorials found that Porta enabled both the tutorial creators and the students to provide more specific, targeted, and actionable feedback about how to improve these tutorials.

These systems together open up possibilities of easing the use of creating quality step-by-step tutorials and user testing existing instructional material in a more systemic and scalable manner.

Chapter 1

Introduction

It can be hard for experts in any field to write high-quality documentation, instructional materials, and step-by-step tutorials. Instructional materials such as these are vital for helping people such as software developers, data scientists, researchers, and system administrators perform complex software-based tasks.

One can either create a written tutorial by painstakingly enumerating all steps, describing shell commands, expected outputs, and side-effects, and taking screenshots to illustrate GUI actions. Or one can demonstrate all of the steps and record a screencast video, but it is tedious to later fix bugs in videos. Neither approach is ideal for creators. Also, neither is ideal for people trying to follow these tutorials: manually-written tutorials may skip over critical details [LGF13], and videos can be difficult to browse and navigate [Kro14, PRHA14].

Additionally, once these tutorials are created, their creators cannot easily get a sense of how people navigate through them or what parts they frequently get stuck on. They also cannot easily predict where new users might struggle [LGF13]. Despite the creator's 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 [NKA01] where experts have trouble relating to what novices might

know and not know.

Thus, learners inevitably struggle because a) Quality tutorials are hard to come by because of the effort to create them. b) Tutorials that do exist have errors or bugs that the creators have no way to diagnose.

These limitations served as our research questions:

1. Can we streamline the process for tutorial creation so that creating tutorial should be as easy as recording a screencast video, but tutorials should offer advantages of text-based formats like easy skimming and copy-paste.

2. 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?

The core technical insight that underpins our solutions to these problems is that the operating system already keeps track of vital filesystem and process-level metadata necessary for creating and evaluating tutorials.

The contributions of this paper are:

1a A new approach to generating mixed-media tutorials by combining screencast video recording with application-agnostic operating-system-wide activity tracing.

1b Torta (**T**ransparent **O**perating-system **R**ecording for **T**utorial **A**cquisition), a prototype system that instantiates this approach for macOS. Torta consists of an operating-system-wide activity recorder, a tutorial editor, and a tutorial viewer that can validate step-by-step progress and even run certain steps.

2a The novel idea of *tutorial profiling* using operating-system-wide activity tracing.

2b Porta (**P**rofiling **O**perating-system **R**ecordings for **T**utorial **A**ssessment), a prototype that instantiates this idea for macOS. Porta consists of an OS-wide

activity recorder, a webpage navigation tracker, and interactive profiling visualizations.

Chapter 2

Related Work

2.1 Operating-System-Wide Activity Tracing

Our approach of tracing activity at the operating-system level was inspired by related work in OS-level tracing for user-facing tasks.

The most basic form of OS-level tracing consists of recording low-level mouse, keyboard, and multitouch events. Nguyen et al. [NL15] record mouse click and drag events alongside a screencast video to make it possible for viewers to browse through the video by clicking on constituent pixels. DemoWiz [CLD14] captures mouse and keyboard events to augment videos with a visual overlay that aids people in giving live presentations. EverTutor [WCC+14] tracks multitouch events on Android smartphones to help users create step-by-step app tutorials. Graphstract [HT07] tracks mouse and keyboard events and takes mini-screenshots of the areas around those events to generate minimalistic static tutorials.

In terms of higher-level OS activity tracing, KarDo [KK10] records GUI traces using the Windows Accessibility API and algorithmically generalizes those traces so that they can be replayed on other machines to replicate user actions and reproduce GUI bugs. DejaView [LBP+07] augments GUI activity tracing with fine-grained

checkpointing of filesystem and OS environment state so that a user can browse and "time travel" back to earlier system states.

Burrito [GS12] traces GUI activity, shell commands, and filesystem modifications, but it was designed to help computational scientists manage their own workflows, not to create tutorials. Thus, Burrito does not include a screencast recorder, video segmenter, tutorial editor, or tutorial viewer/player.

In sum, while these tools employ similar OS-level tracing techniques as our systems, they are meant for tasks such as automation, OS state replication, and scientific workflow management, and thus lack design affordances for automatic tutorial generation and tutorial profiling.

2.2 Related work for Torta

In addition to operating system monitoring, Torta builds upon ideas in three prior lines of work: mixed-media tutorials, generating tutorials by demonstration

2.2.1 Mixed-Media Tutorials that Combine Text and Video

The design of the Torta tutorial format was directly inspired by prior work on *mixed-media tutorials* that combine static (text+image) and video modalities. These tutorials are formatted as a series of steps on a webpage with a mix of textual exposition and embedded mini-videos at each step. Chi et al. [CAR+12] performed a comparative study of static, video, and mixed-media tutorials for image manipulation tasks and discovered that users found mixed-media tutorials the easiest to follow and the least error-prone. From this study they proposed four design guidelines for mixed-media tutorials, which Torta follows: 1) *scannable steps*: Torta segments the creator's screencast recording into steps based on active GUI windows, executed commands, and text file edits, which facilitates scanning, 2) *small but legible videos*: Torta crops each video segment to include only the active GUI window, which emphasizes the

most relevant portions at each step, 3) *visualize mouse movement*, and 4) *give control to the user*: Torta tutorials are hierarchical, so more advanced users can hide specific steps while novices can expand to see more details.

Inspired by this format, the Video Digests [PRHA14] and VideoDoc [Kro14] systems semi-automatically generate mixed-media tutorials from existing videos that contain time-aligned transcripts. Although these were both designed for reformatting lecture and talk videos, they could in principle be repurposed for software tutorial videos as well.

In contrast to these projects, which aim to reformat existing video tutorials, Torta allows users to make mixed-media tutorials from scratch by recording a demonstration of their actions. Since Torta traces OS-level metadata during the user demonstration, it can automatically provide more specific details about software tutorial steps than text transcripts or crowd workers' annotations can.

2.2.2 Generating Tutorials by User Demonstration

Torta's approach of generating tutorials via user demonstration of actions was inspired by prior systems that operated in a similar way within specific software applications.

Image editing applications have been the most popular substrates for such demonstration-based tutorial generators. Grabler et al. built a system that lets users generate photo editing tutorials by augmenting the GIMP image editor to record application and UI state [GAL+09]. Chronicle [GMF10] similarly traces a detailed workflow history within Paint.NET, another image editing application, to facilitate tutorial creation and replay. This was later turned into the Autodesk Screencast [Aut17] product, which records videos, metadata across supported Autodesk applications, app switching and mouse/keyboard events, and provides a tutorial editor similar to Torta's. Lafreniere et al. [LGMF14] generated tutorials by instrumenting Pixlr, an online im-

age editing app, to collect usage logs alongside a screencast video. MixT [CAR+12] creates mixed-media tutorials by capturing screencast videos alongside recorded application state and command logs within Adobe Photoshop.

Other tools in this space also operate within specific apps or preset collections of apps. For instance, ActionShot [LNL+10] augments a web browser to record a rich history consisting of in-browser actions and form entries, which can be used to make tutorials for complex web-based tasks. DocWizards [BCLO05] instruments the Eclipse IDE to let people generate tutorials for software development within that IDE. InterTwine [FLCT14] tracks user actions in both the Firefox browser and the GIMP image editor to create tutorials that span those two applications.

Torta differentiates itself by working across arbitrary sets of macOS applications, whereas these prior tutorial generators operate either wholly within a single application or on a small fixed set of apps. Torta's design trades granularity for generality – since it aims to be generally applicable across all kinds of command-line and GUI apps, it cannot do fine-grained tracing within specific apps like these related projects can do.

2.3 Related work for Porta

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

2.3.1 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 [HHWM92] 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 [ACF+09]. Patina [MGF13] 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 [KGC+14] 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 [DHK16] and ZIPT [DHF+17] 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 [GKM82, SE94] 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 [LBM14] improves on this idea by continually running user code and displaying always-on profile visualizations in the margins of the code editor. Bifröst [MDW+17] 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 novel idea of building usage profiles for *software tutorials* rather than for specific pieces of soft-

ware. 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.

2.3.2 Multi-Application Information Scent

The profiling sidebar that Porta displays over webpages provides information scent [Pir07] 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 [GM09] and HyperSource [HDC11] 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 [FLCT14] 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.

2.3.3 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 [CKW12] and Tagged-Comments [BDL+14] 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 [LGF13] 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.

Chapter 3

Torta: Generating Software Tutorials Using Operating System Wide Activity Tracing

3.1 Introduction

In this chapter, we will look at generating mixed-media tutorials by combining screencast video recording with application-agnostic operating-system-wide activity tracing. Torta is a prototype system that instantiates this approach for macOS. Torta consists of an operating-system-wide activity recorder, a tutorial editor, and a tutorial viewer that can validate step-by-step progress and even run certain steps.

This chapter will look at the formative study that inspired our design goals, the system design and it's evaluation.

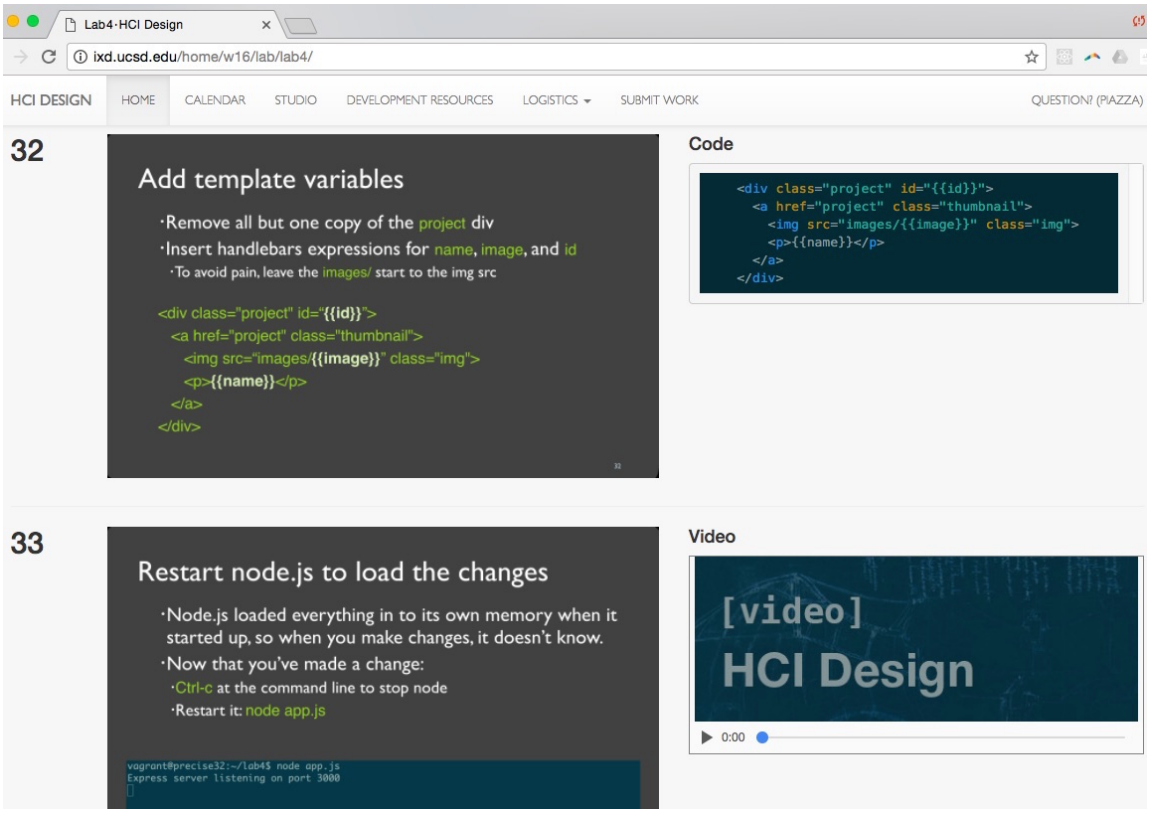

Figure 3.1: A mixed-media tutorial for web programming that was manually created for an HCI course [Kle17]. Each tutorial is up to 100 steps consisting of PowerPoint slides mixed with code snippets and videos.

3.2 Formative Interviews and Design Goals

To discover the challenges faced by people who manually create mixed-media software tutorials, we interviewed four teaching assistants responsible for creating web programming tutorials for an introductory HCI course [Kle17].

In this course, students build full-stack web applications with a mix of tools such as Git for version control, Node.js for server-side development, Handlebars for templating, Bootstrap for responsive CSS, and Heroku for live deployment to the web. Students come into the course from a diverse variety of majors and widely varying levels of prior programming experience, so the staff teaches weekly programming

tutorials to get everyone acquainted with the mechanics of web development (e.g., setup, coding, testing, online deployment). Since these tutorials must coordinate across multiple command-line and GUI applications, they are representative of the sorts of tutorials that we would like Torta to automatically generate.

Each lesson is a webpage containing a mixed-media tutorial interspersing PowerPoint slides, video clips, and command/code snippets. Everything is created manually: The staff first makes a PowerPoint deck (usually around 100 slides) containing step-by-step instructions, commands to run, code to write/modify, and screenshots showing expected visual outputs. They export the slides as images to embed within a webpage and then supplement it with screencast videos and commands/code that students should copy-and-paste into their terminals. From our interviews with teaching assistants, we found three main bottlenecks in creating these tutorials and deploying them during in-class lab sessions:

PowerPoint slides versus screencast videos: Students greatly preferred reading the PowerPoint slides since those were easily skimmable, but the staff found them far more tedious to create since they needed to first demonstrate their actions and then manually copy-and-paste all commands, code, expected outputs, and screenshots into the slides. Also, sometimes the slides did not go into enough detail or skipped steps due to staff oversight or simply lack of prep time. In contrast, screencast videos were much easier for staff to record and contain all necessary details, but were harder for students to browse. The staff struck a compromise by placing slides and videos side by side on the webpage, using videos to showcase dynamic events such as GUI interactions and web animations.

Slides, videos, and code are disconnected: Besides slides and videos, the staff also embedded snippets of code and commands into tutorial webpages. They did this because students found it hard to copy-and-paste directly from PowerPoint slides due to syntax-breaking formatting issues (e.g., bad line breaks, smart quotes, special characters, incomplete code due to lack of space on slides), and it is impossible

to copy from videos. Additionally, the staff maintained a GitHub repository of skeleton starter code and helper scripts for students to build upon when following these tutorials. This heterogeneous setup meant that when the staff created or updated each tutorial, they had to manually keep four disparate data sources all updated and in sync: PowerPoint slides, videos, command/code snippets, and the GitHub repository of skeleton code; these disconnects led to numerous bugs in tutorials.

Hard for students to validate progress: When working through tutorials in class, students were anxious about whether they were following certain steps properly since the PowerPoint slides did not always specify the expected effects of each step, and videos were not available for all steps. Many effects were not immediately visible on-screen, such as the results of running a Heroku configuration command. Even worse, when a student does not follow a step properly, everything may still appear to work, but subtle errors silently propagate and manifest in later steps with unrelated error messages. These problems arise because students cannot easily check their progress. The staff ended up dealing with this by writing *validation scripts* for each tutorial. When a student runs a validation script, it checks that their filesystem state, environment variables, and current directory are what the tutorial expects; otherwise it prints a targeted error message.

These bottlenecks inspired a set of design goals for Torta:

- **D1**: Creating a tutorial should be as easy as recording a screencast video, but tutorials should offer advantages of text-based formats like easy skimming and copy-paste.
- **D2**: A tutorial should automatically encapsulate videos, textual exposition, code examples, and terminal commands together into one package instead of in disconnected silos.
- **D3**: Users should be able to view the tutorial at varying levels of detail to accommodate their own expertise level.

- **D4**: Users should be able to incrementally validate their progress as they follow each step of the tutorial.

3.3 The Design and Implementation of Torta

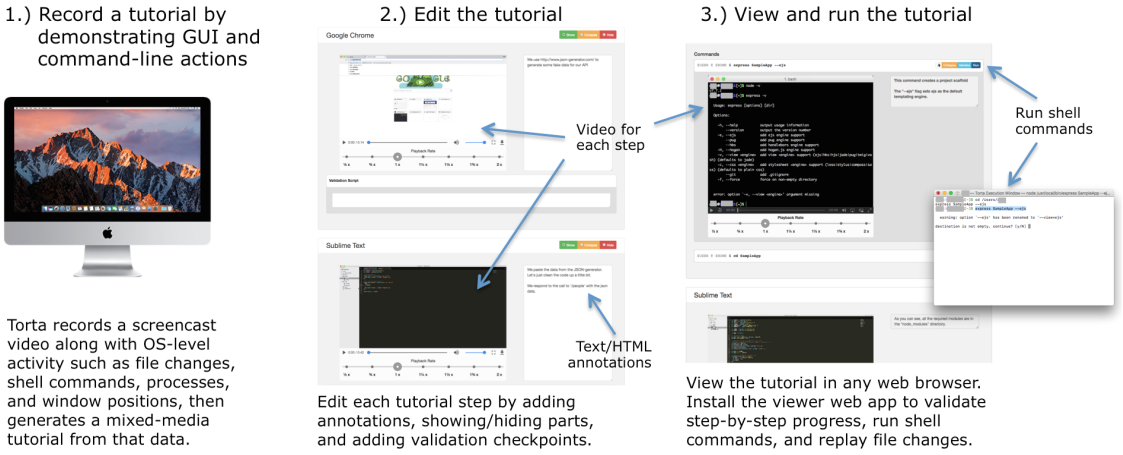

Figure 3.2: Torta allows macOS users to create mixed-media tutorials by demonstration, and then edit, view, and run those tutorials in a web browser.

Torta consists of three components: a tutorial recorder, editor, and viewer (Figure 3.2). We now describe each in turn.

3.3.1 Tutorial Recorder

Torta's recorder allows the user to record a tutorial just as easily as recording a screencast video (Design Goal D1). Our prototype is implemented for macOS using AppleScript, Python, Bash, and DTrace [CSL04] scripts to perform OS-level activity tracing. It should be straightforward to port this OS-wide tracing-based approach to other operating systems.

When the user wants to start recording a tutorial, they activate Torta by running a terminal command, which immediately launches a set of activity tracers. The user

then records their tutorial by simply demonstrating actions on their computer, and the tracers log the following data in the background:

- **Screencast video recorder**: Torta uses Apple's built-in Quicktime app to record a standard full-screen screencast video with audio narration and mouse clicks visualized.
- **Foreground GUI window monitor**: The position and dimensions of the user's current foreground GUI window are logged once per second, along with the process ID of the program that owns the current foreground window.
- **Keystroke logger**: All user keystrokes are logged.
- **Shell command logger**: The contents of all terminal commands run in any shell are logged and timestamped. The current working directory, username, and environment variables used for running each command are also logged. Our current logger works for Bash (the default on macOS) and Zsh, but can be easily extended to other custom shells.
- **Filesystem activity tracer**: Torta uses DTrace [CSL04] to record a subset of system calls that access the filesystem. Specifically, it logs the timestamps, owner process IDs, and parameters of the following filesystem-related system calls: `open()`, `write()`, `close()`, `rename()`, and `unlink()` (for opening, writing to, closing, renaming, and deleting files, respectively). Torta makes a timestamped backup copy of each affected file after the respective system call is run. This feature is useful for saving all versions of files that users edit within interactive applications such as text editors, IDEs, or Photoshop: Each time the user presses "Save" within the app, a `write()` system call occurs, and Torta saves a backup copy, which lets it later display diffs.
- **OS process tree logger**: Torta logs the command names, start/end timestamps, process IDs (PIDs), and parent process IDs (PPIDs) of all OS processes

launched after the user activates Torta. This log serves two purposes: First, it filters the system call trace (see above) to consider only processes that launched *after* the user activated Torta, which eliminates the noise from dozens of irrelevant system-wide processes previously running on the user's machine. Second, it is necessary for linking the system call trace to foreground GUI windows. Here is why: Many interactive apps adopt a multi-process model for robustness. For instance, Google Chrome launches one OS process per browser tab, and text editors such as Sublime Text launch one OS process per text editor tab along with a separate process for the GUI. Thus, the process that owns the Sublime Text foreground GUI window is *not* the process that makes the `write()` system calls to save the user's files. Torta can use the OS process tree of PIDs and PPIDs to link Sublime Text's user-initiated file save events with its GUI window, since they are owned by sibling processes.

After the user finishes recording their demonstration and shuts down Torta, it automatically creates a *mixed-media tutorial* by post-processing and combining the recorded data into self-contained package that contains all traces, segmented videos, and saved file versions (Design Goal D2). As shown in Figure 3.3, a Torta-generated tutorial has a hierarchical structure that aims to follow the design guidelines of Chi et al. [CAR+12]:

Top-level steps – foreground GUI windows: A Torta tutorial is an ordered list of top-level steps. Each step spans the duration of one foreground GUI window. Torta uses FFmpeg [ffm17] to split the screencast video into one mini-video per foreground window duration and crops those videos to show only the foreground window. We felt that foreground windows were the most natural step boundaries for these kinds of software tutorials, since users often perform a set of actions within one window (e.g., an IDE) and then switch to another window (e.g., Photoshop) to perform the next set.

Each step is rendered as a mini-video along with a *filesystem tree* showing which

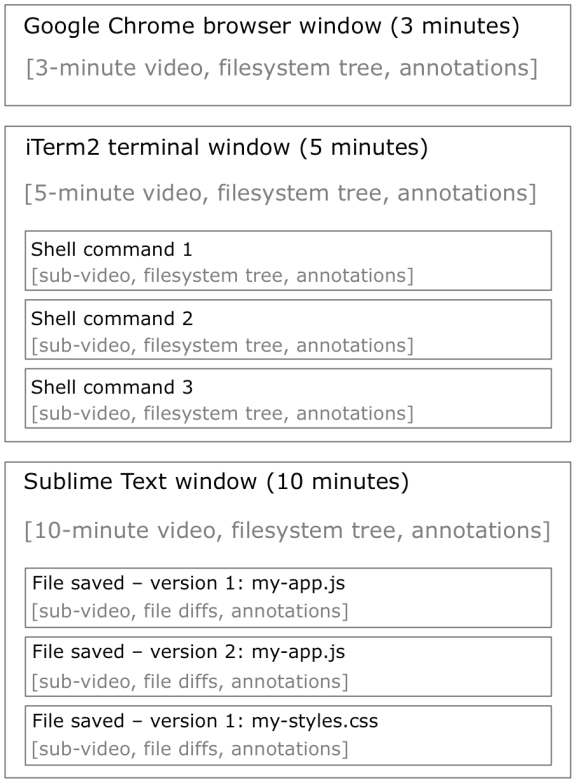

Figure 3.3: Example structure of a mixed-media tutorial generated by Torta. Each of the three steps represents a foreground GUI window duration. There are three sub-steps within the terminal and IDE windows.

files were added, deleted, renamed, and modified by processes associated with the foreground GUI window during that step (Figure 3.5). We chose to visualize filesystem changes since those represent the persistent effects of user actions within an application. Regardless of what kind of app the user is running, if some action has a lasting effect on their computer, it will likely manifest in the filesystem.

Sub-steps: Torta further splits each top-level step into sub-steps based on two common kinds of user actions (Figure 3.3):

- ***Shell commands***: If the user runs multiple shell commands within the duration of one foreground window (usually some kind of terminal app), Torta splits that step into one sub-step for each command. Each sub-step is shown as a

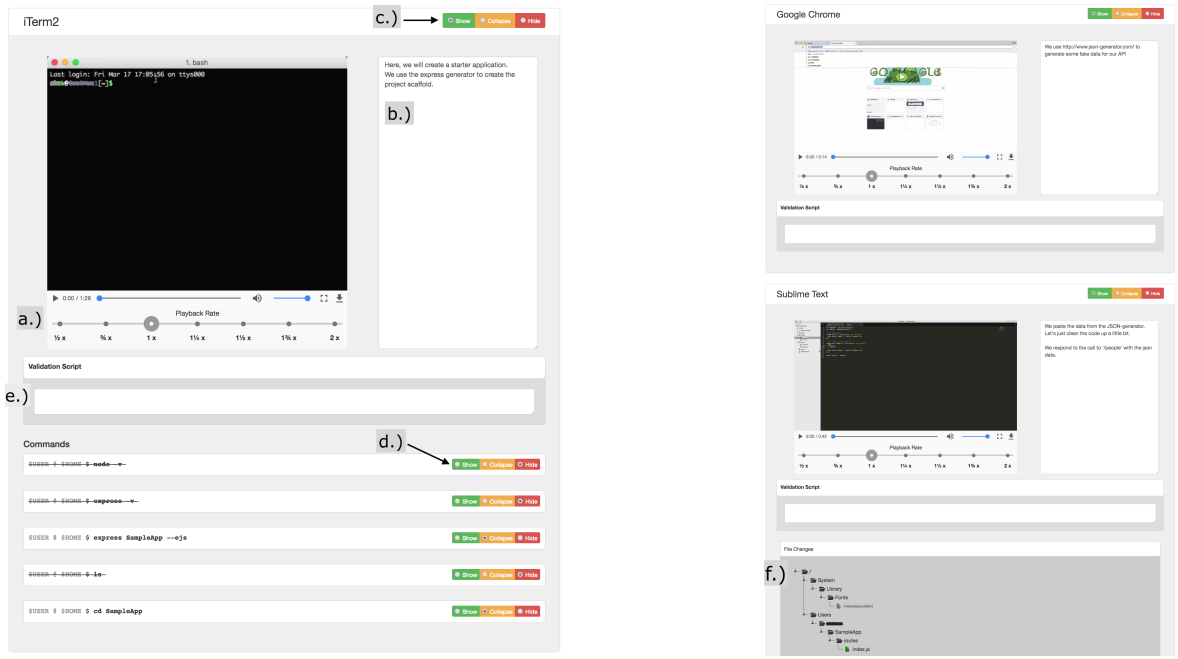

Figure 3.4: Zoomed-in screenshots of Torta's tutorial editor showing three steps (i.e., foreground windows): iTerm2, Google Chrome, and Sublime Text. Each step contains: a.) Video player with playback speed adjuster, b.) Text annotation box, c.) Toggle to show/collapse/hide this entire step, d.) Toggle to show/collapse/hide each sub-step (here the hidden shell command sub-steps are crossed-out), e.) Validation script, f.) Filesystem tree (see Figure 3.5).

mini-video spanning the duration of only that command, the text of that command, its current working directory, environment variables, and a filesystem tree showing what files that command added, deleted, renamed, and modified.

- ***File saves***: When the foreground window is an interactive app such as an IDE, web browser, or image editor, the user may be editing files and periodically saving their progress to disk. Torta splits each step into sub-steps based on file save events, treating saves like user-defined checkpoints in the tutorial. Again, each sub-step gets its own mini-video. If the saved file is plain text, Torta also shows the diffs between the current and previously-saved versions, which

is useful for showing edits in code and configuration files.

3.3.2 Tutorial Editor

The mixed-media tutorial that Torta automatically generates from the user's demonstration is already complete and ready to view on the web. One can think of it as a screencast video that is segmented and enhanced with OS-level trace data. However, it can be hard for users to record a pristine, error-free video in one take. Furthermore, users also want to augment tutorials with textual annotations and other customizations. To fulfill these needs, Torta provides a tutorial editor, which renders the tutorial just as the viewer would see it but adds extra controls for the following actions (Figure 3.4):

Adding text annotations to steps/sub-steps: The user should already provide audio narration when recording their demonstration, which will show up in the screencast video. The editor also lets them add Markdown-based rich-text annotations next to the segmented video for each step/sub-step.

Hiding steps/sub-steps: The user can hide any step/sub-step from the viewer to eliminate mistakes or redundancies (effectively deleting them from the edited tutorial). If the user hides a step that is in between two steps that belong to the same application, then those two surrounding steps get merged into one. This happens when, say, the user is in an IDE, then switches to a web browser to look up something quickly, then switches back to the IDE. If the user hides the web browser step because they deem it irrelevant for the tutorial, then the two IDE steps get merged together as one step in the viewer. Torta does not support post-hoc re-recording of steps in the editor. A workaround is to record an entire session even with errors included and then hide erroneous steps using the editor.

Collapsing steps/sub-steps: If the user deems certain steps or sub-steps to be less important for the tutorial, they can show them in a collapsed form. The viewer

will see those steps as a collapsed summary but can manually un-collapse them to dive into details. Torta displays compact summaries so that viewers can more easily skim step contents (e.g., “Photoshop window active for 2 minutes, modified 3 files").

Torta implements heuristics to automatically collapse certain steps/sub-steps that are likely to be less important to the tutorial. For instance, if a shell command does not make any changes to the filesystem (e.g., `ls` or `git status`), it is collapsed by default since the user was probably checking their setup before proceeding to the next step. Also, if any GUI window was in the foreground for less than 5 seconds, had less than 10 user keystrokes, and did not modify the filesystem, then its step is also collapsed by default. This filters out “flickers" where the user switches between windows momentarily to quickly check something before the next step.

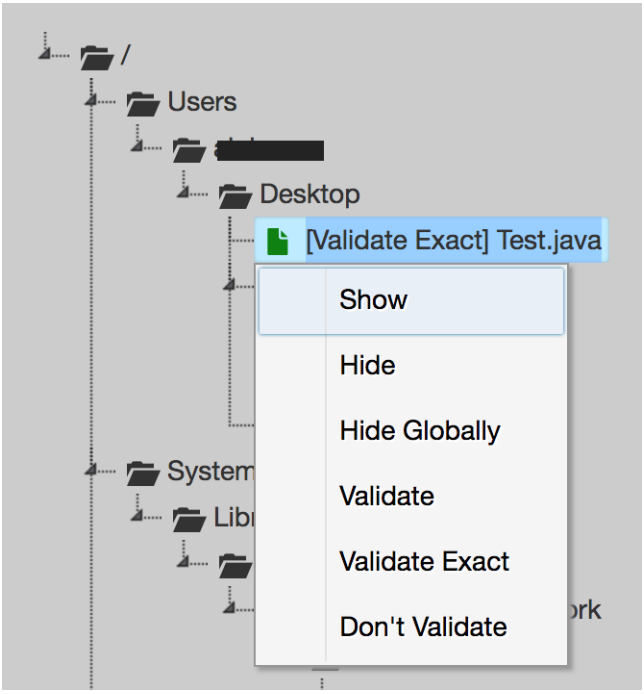

Figure 3.5: Each file-modifying tutorial step displays a filesystem tree of all files affected by running that step. In the editor UI, the user can right-click to show/hide files and to mark for validation.

Collapsing filesystem tree components: Recall that Torta displays a filesystem tree within each step and sub-step that modifies the filesystem (Figure 3.5). However, during pilot testing we noticed that some commands (e.g., `git clone`) can affect hundreds of files, so their trees are extremely large. To reduce visual overload, the user can collapse tree nodes to hide and summarize their sub-trees. For instance, the user can collapse a `.git/` sub-directory to see a summary like "100 files added and 15 files modified in `.git/`." Just as with collapsed steps, the viewer can un-collapse tree nodes to see more details on demand. The user can also choose "Hide Globally" to hide a particular file/directory across all tutorial steps.

Adding validation: The editor provides two ways to specify how people (i.e., tutorial *consumers*) can validate progress at each step as they are following the tutorial (Design Goal D4):

1. Marking files to validate: The user can mark each file in the filesystem tree of a step/sub-step as "Validate", "Validate Exact," or "Don't Validate" (Figure 3.5). If the user marks a directory, everything within it also gets marked with that label. "Validate" means that Torta should check that the consumer's file gets altered in the way that this step specifies (e.g., modified or renamed), and "Validate Exact" means that the new contents of the file should also exactly match the saved version bundled in the tutorial package. For example, in a step where the consumer is supposed to add their username to a section within a configuration file, that file should be marked as "Validate" to check that it has been modified, but not "Validate Exact" since everyone's username will be different.

2. Writing validation scripts: File-based validation handles the most common uses, but if tutorial creators want more flexibility, they can write a validation script for each step/sub-step (Figure 3.4e). This is a Bash script that will run on the consumer's machine to check that their OS state is as expected.

This feature is similar in spirit to the step-level validation features offered by tutorial systems for other domains [FGF11, PDL+11].

After the user finishes editing the tutorial, they can publish it as a webpage or send the self-contained package to viewers.

3.3.3 Tutorial Viewer

Since Torta tutorials are ordinary webpages, they can be viewed in any browser. Each tutorial initially loads with certain steps/sub-steps collapsed, certain file tree nodes collapsed, and each video playing at the speed pre-set by the creator. However, the user can adjust any of those settings. In addition, they can click on any file in the tutorial and view/download the version of that file present during that respective step (all versions are stored in the package). This ability to selectively hide and show details was inspired by a challenge discovered during formative interviews: Students preferred seeing varying levels of detail depending on their expertise level (Design Goal D3). It is hard to achieve this flexibility with raw screencast videos or PowerPoint slides.

To make tutorials more readable, Torta canonicalizes all file paths within command invocations and filesystem trees. For instance, when Alice creates a tutorial, many of her file paths will contain `/home/alice` if they are within her home directory. But when Bob is viewing the tutorial, he would prefer to see paths starting with `/home/bob` instead of `/home/alice`. Torta canonicalizes paths by replacing the creator's home directory with the `$HOME` variable. Additionally, the creator can use the tutorial editor to specify other path variables to replace. One use case is specifying a `$PROJECT_ROOT` directory where all files within a project should live. The tutorial viewer prompts the user to enter their own preferred values for all of these variables and rewrites all paths within the webpage accordingly. Note that `$HOME` and other environment variables are automatically set if the tutorial is loaded from the user's machine rather than viewed on the web.

If the user downloads the tutorial to their macOS machine and loads it via the

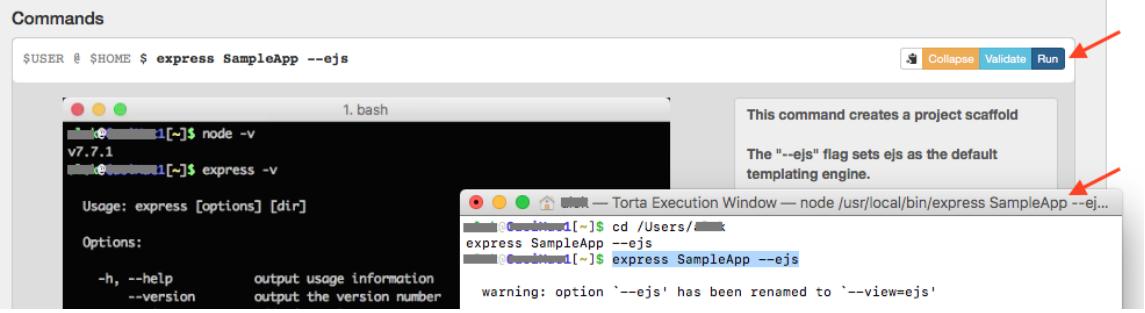

Figure 3.6: Each step and sub-step contains “Validate" and “Run" buttons on the upper right. Here when the user clicks on “Run" for a shell command sub-step, Torta runs that commands in a new terminal window.

Torta viewer web app on localhost, then they can access two additional features as shown in Figure 3.6:

1. Validating step-by-step progress: After manually performing the actions specified by a particular step/sub-step, the user can click the “validate" button alongside its video. Torta will check that the affected files on the user's local filesystem have been modified in the ways that the creator originally expected (i.e., specified via “validate" and “validate exact" labels in the filesystem tree) and also run the validation script if it exists. Then it prompts the user if there are errors and offers to overwrite any mismatched files with the versions from the tutorial package if the user wishes. This capability lets the user check that they are properly following along with each step of the tutorial and to catch bugs earlier (Design Goal D4).

2. Automatically running steps: The user can click the “run" button next to each step/sub-step to have Torta automatically run that step for them. For a shell command, Torta launches a terminal app on the user's machine and runs the command from that terminal after setting the proper working directory and environment variables. For a step involving a GUI application, Torta does not try to replay GUI actions but rather simply mutates the user's filesystem in the way that has been prescribed by that step. Although this approach is not always guaranteed to be fully faithful to that step's actions, in practice it works well in some cases since

	First	Tutorial Topic	GUI Apps	Command-line Apps	Torta Time (recorder+editor)	# Torta Steps (before edits)	GDocs Time	# GDocs Steps
S1	GDoc	HTML and CSS web design	Finder, Sublime Text, iTerm2, Google Chrome	node, touch	14m (8+6)	8 (11)	30m	7
S2	GDoc	Node.js server setup and API endpoint creation	Finder, Sublime, iTerm2, Chrome, Postman	npm, express, node, touch	15m (7+8)	12 (14)	24m	†
S3	GDoc	JavaScript frontend web dev	Finder, Sublime, iTerm2,Chrome	python	17m (12+5)	12 (16)	32m	8
S4	GDoc	Data science: neural nets w/ Keras and TensorFlow	Finder, Sublime, iTerm2	python, pip	19m (11+8)	3 (8)	38m	3
S5	GDoc	Data science: linear regression w/ scikit-learn	Finder, Sublime, iTerm2, Chrome	python, pip, brew	15m (12+3)	5 (8)	43m	5
S6	Torta	Go language toolchain install and setup	Finder, Sublime, iTerm2	gcc, make, cat, golang, brew	31m (21+10)	13 (18)	32m	10
S7	Torta	Java singly-linked list	Finder, Netbeans, iTerm2	javac, java	25m (15+10)	6 (7)	28m	5
S8	Torta	C doubly-linked list	Finder, iTerm2, Chrome	gcc, vim	27m (22+5)	6 (10)	24m	4
S9	Torta	Python list comprehensions	Finder, PyCharm, iTerm2	python	16m (10+6)	4 (7)	23m	5
S10	Torta	C binary search tree	Finder, iTerm2, Chrome	gcc, vim	29m (21+8)	10 (15)	30m	10

Table 3.1: Tutorial creator study results, showing subject IDs, which tool they used first, summary of their tutorial, time in each tool, and the numbers of steps in Torta and GDoc tutorials († did not explicitly denote steps in GDocs). All times reported in minutes, with Torta split into recorder+editor times.

the persistent effects of a GUI application usually manifest in the filesystem. For instance, if someone demonstrates how to use a GUI to customize the configuration of a complex interactive application, the effects of that customization may show up as changes to some config file. When the user hits “run" on that step, Torta simply copies over the updated version of that config file.

3.4 Exploratory User Studies

As a first pass at illustrating Torta’s capabilities, we compared users’ experiences of both creating and consuming Torta tutorials to doing so with manually-written tutorials. We chose to initially compare Torta to manually-written text+screenshot tutorials since those are now ubiquitous on the web in the form of technical blog posts, documentation websites, PowerPoint presentations, course lecture notes, Q&A and forum posts, and (electronic+paper) books. Note that although we compared with written tutorials for this exploratory study, a more rigorous controlled study would have also compared Torta to recording and editing screencast videos, since that is a

closely-related and also-ubiquitous format for tutorials.

To cover the two target audiences for Torta, we ran two exploratory user studies: 1) A study on tutorial creators, which tests Torta's recorder and editor components, and 2) a study on tutorial consumers, which tests Torta's viewer app.

3.4.1 Tutorial Creator User Study

First we compared the user experience of creating a tutorial with Torta versus manually writing a tutorial in Google Docs. We chose Google Docs since it is a convenient way for someone to quickly create a written tutorial; it supports rich text formatting, copy-and-paste of screenshot images, and does not require specialized knowledge of HTML or other markup languages. (Microsoft Word would have worked just as well.)

Procedure: We recruited 10 graduate students who have served as teaching assistants (TAs) for computer science courses to each perform a 1.5-hour lab study using both Torta and Google Docs on a 21.5" iMac. We told each subject to create a multi-application software tutorial for a relevant topic from a class that they have TA'ed. To counterbalance tool order, we had five subjects first spend up to 40 minutes creating their tutorial in Google Docs, then try to re-create that *same tutorial* in Torta. We had the other five subjects use Torta first, and then re-create the same tutorial in Google Docs.

Right before each subject used Google Docs, we encouraged them to design a well-structured step-by-step tutorial with a mix of text, screenshots, and formatted code/command snippets in monospaced font to emulate a technical blog. Right before each subject used Torta, we gave them a five-minute tutorial (a "Tortorial") on Torta's recorder and editor UIs.

We spent the final 10–20 minutes of each session conducting a semi-structured interview where we had the subject compare their experiences using Torta and Google

Docs then self-assess the quality of the tutorials they created in both tools.

Results: Table 3.3.3 summarizes the generated tutorials. All involved multiple GUI and command-line apps such as IDEs, compilers, build tools, package managers, and webservers. All subjects used the Torta editor to eliminate a few steps that arose from errors in their recording (the "before edits" entries in Table 3.3.3). They also collapsed "boring-looking" steps such as restarting the Node.js server repeatedly. However, they did not write many textual annotations due to lack of time and because they had already recorded voice narration in videos.

During the post-study interviews, all 10 subjects self-reported that they preferred using Torta over Google Docs (GDocs). They also all self-reported that they felt their Torta-generated tutorials were better organized and higher quality. Multiple subjects mentioned the following points of contrast:

- *Torta eliminates context switching*: When using GDocs, subjects often had to perform a step, pause, switch to write their instructions and paste screenshots in the doc, then switch back and forth; they felt this process was inefficient. In contrast, Torta recorded seamlessly without interruption.
- *No need to manually write/paste commands, code diffs, and file changes in Torta*: In GDocs, users had to manually write (or copy-and-paste) the commands, code changes, and file changes for each step, whereas Torta automatically captures all of those details. Torta users also liked how each change was also captured in a short video snippet.
- *Taking/organizing screenshots is cumbersome in GDocs*: All 10 subjects took screenshots of their computer activity when creating their GDocs tutorial. They found it awkward to manage a stockpile of similarly-named screenshot image files on their desktop. And they had to often browse through a pile of files, crop them properly, and copy them into the doc. With Torta, they could demonstrate their actions and have the screencast video recorded automati-

cally.

- *Demonstrating GUI actions was much easier on video*: Subjects who heavily used GUI tools such as the Postman [pos17] API tester app (S2) found it much easier to demonstrate how to use the tool in a video rather than taking static screenshots and writing about user flow on GDocs.
- *Think-aloud in Torta felt more natural than writing text in GDocs*: All subjects preferred to vocalize their thought process as they demonstrated actions within Torta. They could use more casual, extemporaneous language rather than feeling obligated to write more formally in a GDoc.
- *Torta allows highlighting code and commands to verbally explain them*: Several subjects found it intuitive to highlight parts of code and commands while verbally explaining them in Torta. To do the same thing in GDocs, they needed to paste a snippet into the doc and then describe it.

Note that all these limitations of written tutorials are not specific to Google Docs; related tools likely face similar issues.

Although we did not directly compare Torta to screencast videos for this study, several creators mentioned the similarities between Torta and screencast recording. For instance, S5 said, *"This [Torta] isn't any different from recording a screencast and I can also do editing, annotation and validation."* And S2 mentioned, *"I think I would use this over recording a screencast despite the additional processing time since the editor allows easier basic editing, like dropping steps, which is easier than using a full video editor."*

Subjects also conveyed perceived shortcomings of Torta's creation workflow: It took some a while to get used to Torta segmenting videos based on OS events such as GUI window switches, so they had to learn to finish a full sentence of narration before switching in order to avoid awkward audio cuts. They wanted to have more diverse format choices than the step-by-step GUI-window-delimited structure that Torta

imposes. They also felt written tutorials were more flexible and less constraining, though they take much more work to make.

Finally, Table 3.3.3 shows that 9 out of 10 subjects created tutorials faster using Torta than Google Docs. For the five who used Torta first, they took around 1.1 times longer to create the GDocs version (the harmonic mean of their time differences); for the five who used GDocs first, they took an average of 2 times longer in GDocs. This speed difference is likely due to them needing to spend time planning out their tutorial's structure during their first attempt, regardless of tool. This phenomenon likely resulted in a significant learning effect since by the time they tried the second condition, they already knew exactly what tutorial they wanted to create. However, even when using Torta first, subjects still found it to be slightly faster. But we do not want to overemphasize these timing numbers because the primary design goal of Torta was not to optimize for tutorial creation speed.

3.4.2 Tutorial Consumer Pilot Study

Although in the prior study we had creators self-assess the quality of their own Torta and GDocs tutorials, we also wanted to get an assessment from the actual target audience: students. Thus, we ran a follow-up pilot study where students followed the tutorials created by the TAs in the prior study and compared their perceptions of Torta vs. GDocs from their perspectives as tutorial consumers.

Procedure: We recruited 6 undergraduate computer science students each for a one-hour lab study. We had 3 subjects try to follow a Torta-generated tutorial; then we showed them the Google Docs version of the *same* tutorial and had them compare and contrast the two formats. We had the other 3 subjects try to follow a Google Docs tutorial, and then had them compare it to its Torta counterpart. We did not have each subject try to follow both tutorials since they would already know how to perform the task after following the first one.

For this study, we picked the Node.js web programming tutorial created by S2 since it was the most complex one. However, two subjects did not have enough technical background to understand it, so instead we gave them the singly-linked list tutorial created by S7 (one used Torta, one used GDocs).

Results: All 6 subjects successfully completed the tutorial tasks in their given format. Both sets of 3 subjects (i.e., those who tried to follow the Torta tutorial then saw the Google Docs version, and vice versa) preferred consuming Torta tutorials over Google Docs for a variety of reasons, including:

- *Torta tutorials were better-structured*: Once subjects got used to Torta's structure, they appreciated its predictability and could skip over parts that did not interest them. In contrast, GDocs does not impose any structure, so tutorials created within it felt more uneven in pace. Subjects also commented that the consistency in Torta's format would be good for a series of related tutorials across an entire course.
- *Torta tutorials more information-dense*: Most subjects commented that Torta tutorials were more information-dense than GDocs since they were narrated by voice and included automatically-traced filesystem and command info. The GDocs versions could not include many details due to lack of time for creators to write out everything explicitly.
- *GUI apps better explained with mini-videos*: All subjects felt that video was a much better way to demonstrate GUI applications such as Postman rather than seeing a series of screenshots in GDocs. They also appreciated each video being short and focused on only one window.
- *Torta provides context behind file diffs*: When using GDocs to create tutorials involving code, most creators ended up simply pasting the new bits of code written in each step into the doc without adequate surrounding context. Thus, subjects were confused about where those pieces of code were supposed to

be placed. In contrast, Torta automatically generates file diffs and shows the original file contents along with mini-videos to give students the proper context.

- *Torta's Validate and Run buttons were popular*: All Torta-using subjects tried the Validate and Run buttons and commented that they seemed very useful. They liked using Validate to avoid minor errors compounding in later steps.

However, one student (who had the most web dev experience) preferred skimming a well-written GDocs tutorial rather than having to play the Torta mini-videos and listen to audio narration at each step. Torta also lets creators write HTML annotations for each step, but due to our user study's short time limit, creators did not write much text in their Torta tutorials.

Finally, although we did not directly compare Torta to raw screencast videos for this study, several subjects implicitly compared Torta with their prior experiences of watching screencasts. For instance, S11 said, *"A lot of times when I'm watching coding videos on YouTube, I wish I would have had a way to copy code and commands. This [Torta] makes that way easier."* S13 said: *"I think cropping videos to the frontmost window is great since it narrows focus to just that window. Many screencasts I watch record the entire screen."* S13 also mentioned: *"I liked Torta breaking the video into steps. On YouTube, video descriptions sometimes have links to different parts of the video but using this [Torta] is much easier because I see the parts of the video already split up."*

3.5 Discussion of Design Space and Limitations

Torta carves out a new point in the design space of tools for generating step-by-step records of app actions (Figure 3.7). In contrast to prior systems that operate within a single application, Torta is designed to be as application-agnostic as possible so that it can work across arbitrary desktop applications. This design decision means

giving up granularity for generality: Torta cannot perform fine-grained domain-specific tracking within any particular application, but rather operates at the level of GUI windows, shell commands, system calls, and filesystem mutations. One future way to bridge this gap is to implement a plug-in system similar to Burrito's [GS12] where users write application-specific tracers to hook into Torta.

On the spectrum of manual to automated running of steps, Torta lies mostly at the manual end. Its primary goal is to generate mixed-media tutorials for people to consume by manually following the steps and customizing based on their needs. In contrast, fully-automated scripting engines serve a different purpose than tutorials: Their goal is to automate a set of repetitive actions, *not* to show people how to manually perform those actions with accompanying pedagogical context. In sum, Torta's output should primarily be thought of as a *rich form of documentation* to help people figure out how to perform tasks for themselves, not as a fully-automated script.

Torta is situated at a point in the design space that makes it well-suited for a range of filesystem-modifying and command-line-heavy software tasks. Even though our original motivation was full-stack web development, Torta is also useful for other complex software development tasks involving more than simply an IDE. For instance, game programming tutorials use an IDE, a 3D level editor, various multimedia editors for assets, and project management tools all tied together by command-line scripts. Low-level systems programming tutorials involve a mix of command-line and GUI tools for introspection, debugging, and performance profiling. Sysadmin (system administration) and DevOps tutorials touch many corners of the operating system at once. Data science tutorials combine multiple programming languages with GUI-based data acquisition/wrangling tools. Finally, scientific researchers hack up ad-hoc workflows that span multiple scripting languages, scientific libraries, and legacy research tools [Guo12], so they can use Torta to generate documentation that can help colleagues reproduce and build upon their computational experiments. How-

ever, Torta is less well-suited for information-foraging-heavy computing tasks such as scholarly research and CSCW-types of communication workflows, since those involve fewer filesystem modifications and shell commands. For those kinds of tasks, Torta simply acts like a screencast recorder with window-based segmentation.

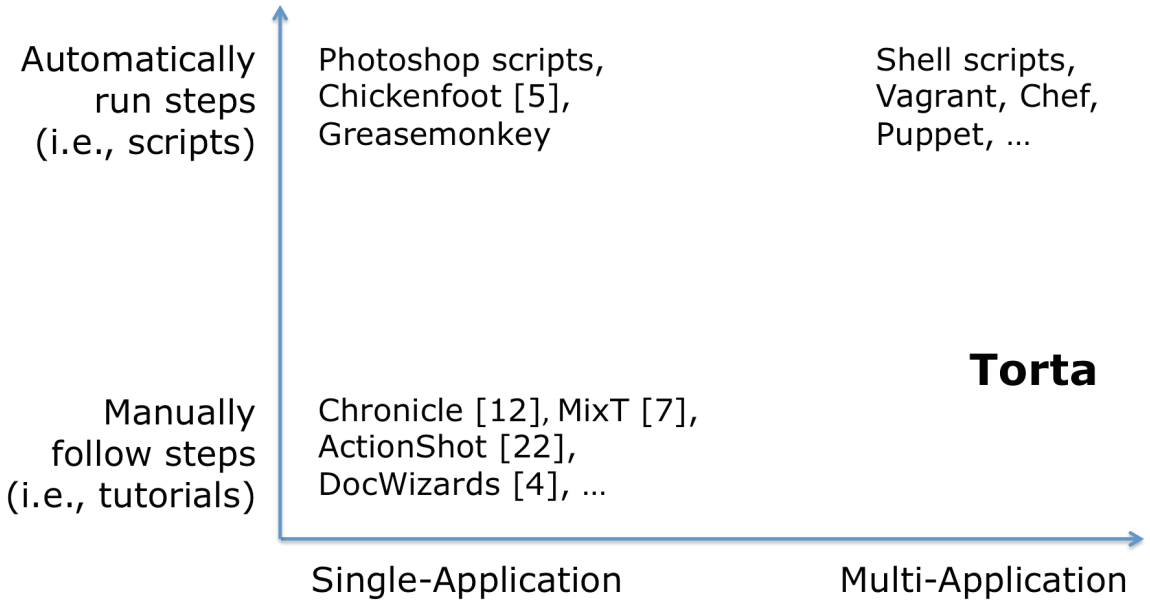

Figure 3.7: The design space of tools for generating step-by-step records of application actions, ranging from those that automatic run (i.e., scripts) to those meant for users to manually follow their steps (i.e., tutorials).

One of Torta's current limitations is that it supports recording and editing only one user demonstration at a time. A future version of the editor could support intelligent merging of multiple demonstrations based on OS activity traces. Another limitation is that Torta does not try to generalize the tutorial's contents: All recorded steps are specific to the creator's single demonstration. Thus, it is up to the creator to manually describe how each step could potentially be generalized or customized by consumers. Again, a more intelligent editor could take multiple demonstrations and semi-automatically infer possible generalizations to make tutorials more robust.

Finally, even though automatically running certain steps can be convenient, we

have purposely not designed the Torta viewer as an fully-automated tutorial runner. Differences between users' OS setups and properties of specific applications make it impossible to automatically run all steps with full accuracy. Thus, we still intend for users to manually follow tutorial steps and only use automatic running as a supplement.

3.6 Acknowledgment

The content from this chapter is borrowed from "Torta: Generating Mixed-Media GUI and Command-Line App Tutorials Using Operating-System-Wide Activity Tracing", Mysore et al. ACM UIST 2017 [MG17]

Chapter 4

Porta: Profiling Software Tutorials Using Operating System Wide Activity Tracing

4.1 Introduction

In this chapter, we will look at the idea of *tutorial profiling* using operating-system-wide activity tracing. Porta is a prototype that instantiates this idea for macOS. Porta consists of an OS-wide activity recorder, a webpage navigation tracker, and interactive profiling visualizations.

This chapter will look at the system design, scope and evaluation of Porta.

4.2 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

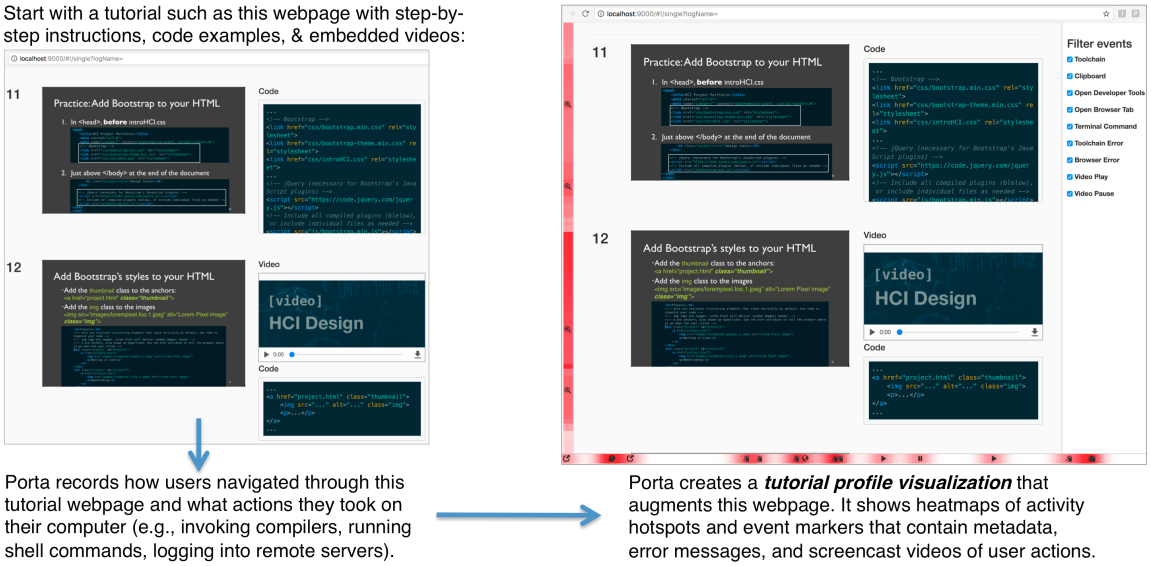

Figure 4.1: Porta provides feedback to tutorial creators about how users navigated through their tutorials and what application errors they encountered.

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.

4.2.1 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 [CSL04] 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 [CSL04] 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 [mim18].

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 [Fox18]. In the future, we can also port the tracer to Windows using its Process Monitor [Rus18].

  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 [MG17] contains similar screencast video and shell command recorders, but all other components described here 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*.

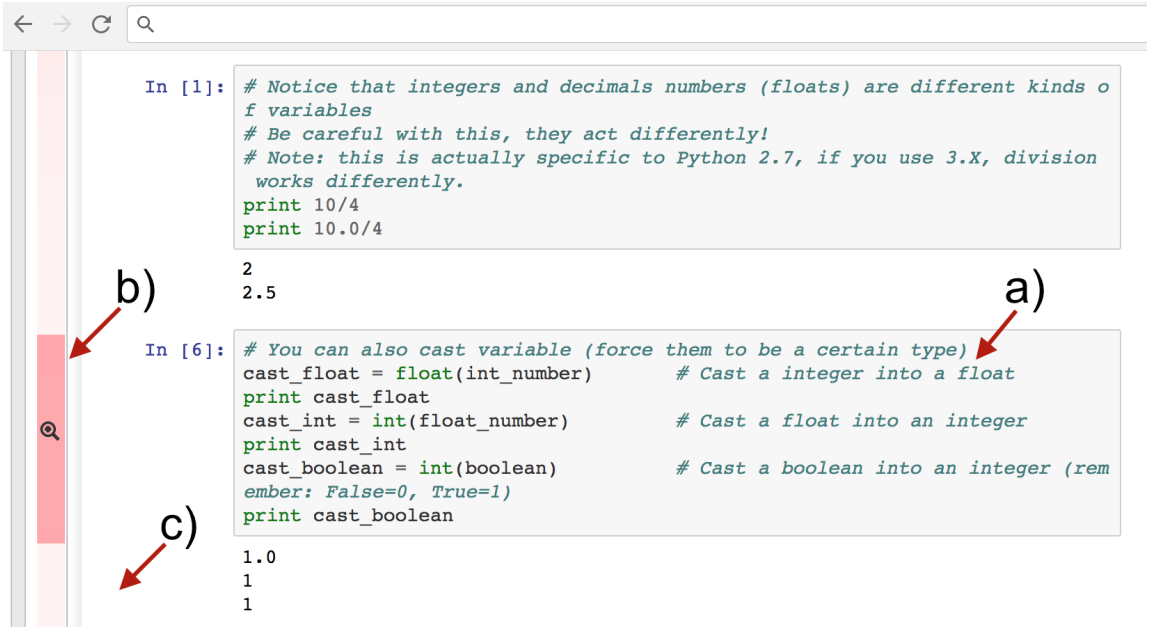

Figure 4.2: Porta uses mouse location as a proxy for where the user's attention is focused. a) If the user hovers over anywhere in this code block element, Porta will record it as being in focus and b) render it as a red hotspot in the sidebar heatmap. c) If the user hovers over an element (e.g., background) that is larger than the viewport, that event is ignored.

4.2.2 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 4.2). This provides a rough indicator of what their attention is focused on at each moment. Ideally

we would gather data from an eyetracker 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 [ful18], Hotjar [hot18], and Mouseflow [mou18].

(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 4.2c). 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 4.2a, 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 [tob11]. 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 [ful18, hot18, mou18]), the main novelty of Porta lies in combining web tracking with the application usage profiler from the prior section to create novel *tutorial profiling visualizations*.

4.2.3 Tutorial Profiling Visualizations

Table 4.1 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.

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

- 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 4.3 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 4.1 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 [GKM82, SE94] 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.

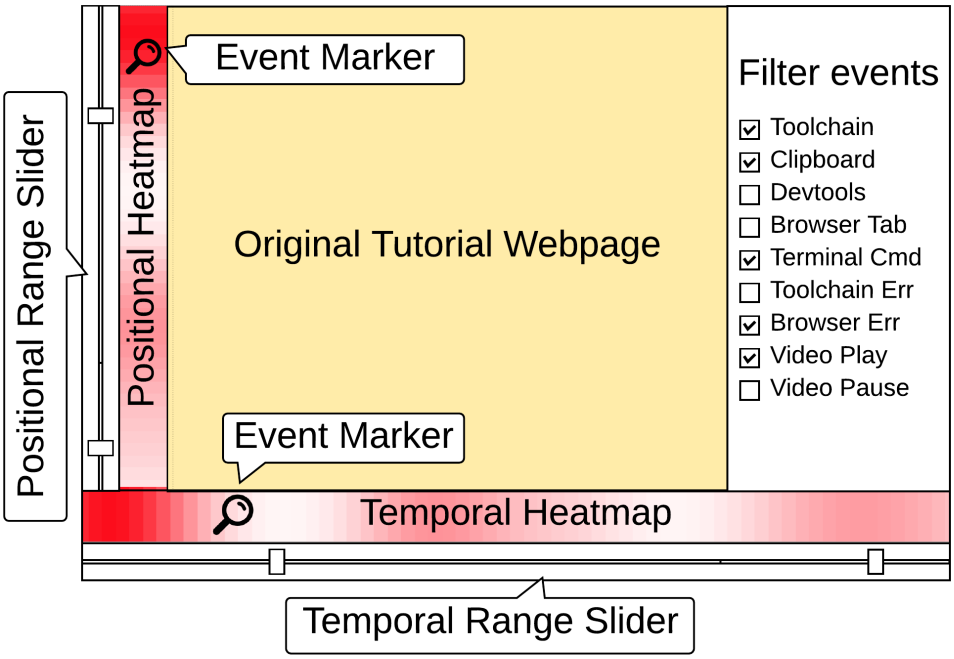

Figure 4.3: Porta uses the trace data collected from test sessions to augment the original tutorial webpage with heatmap visualizations showing positional focus and temporal density of events. Specific occurrences of events show up as clickable markers on both heatmaps.

This heatmap shows the approximate amounts of time that the participant spent on each vertical portion of the webpage throughout the session. Figure 4.4 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

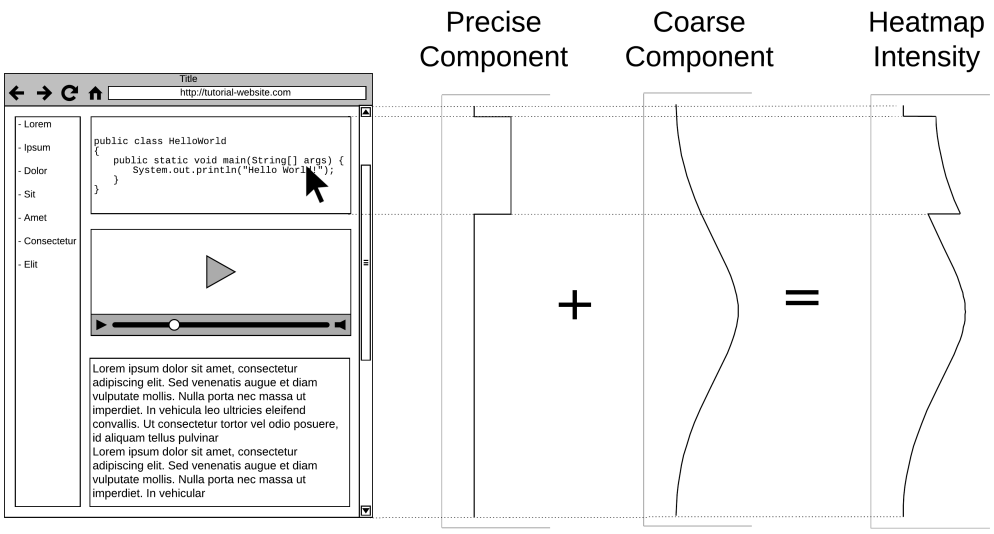

Figure 4.4: The positional heatmap's intensity is a time-weighted sum of precise (mouse hovering over a specific DOM element) and coarse components (Gaussian centered at the middle of the viewport).

on browser scroll position and viewport size. For example, in Figure 4.4, 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 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 4.1 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 4.1 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 4.5a), a pop-up dialog appears to show the details of that event (Figure 4.5b). 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 4.5c). This way, viewers can see the context leading up to the selected event.

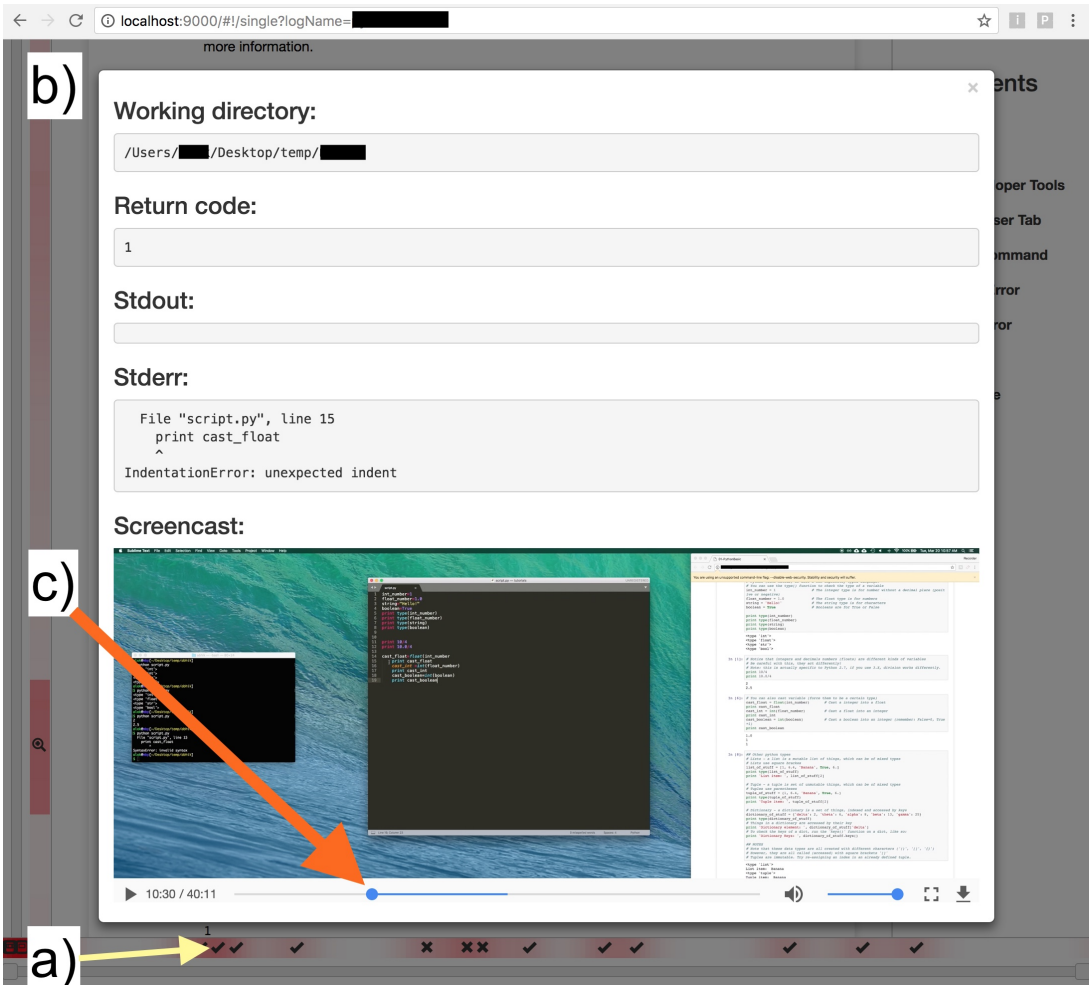

Figure 4.5: Event markers and pop-ups: a) When the user clicks on an event marker on a heatmap, Porta will b) pop up a dialog showing details about that event. c) This dialog also includes a fullscreen video of the test session that starts playing 20 seconds before that event occurred.

It also displays more detailed contextual data depending on event type (Figure 4.5b): 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.

4.3 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:

- **Python**: A primer on basic Python types, control flow, and functions; created for a data science course (~800 words).
- **Git**: Intro. to the Git version control system and GitHub; created as part of a MOOC on introducing scientists to command-line development tools (~3,000 words).
- **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. To get an additional baseline for comparison, we could follow up with a between-subjects study where we show one group a raw screencast video recording of the test session instead of Porta's output. (Note that Porta includes a full screencast recording of the session, but it is segmented based on events.)

4.3.1 Findings from Student User Study

Table 4.2 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 4.2 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 4.3.1 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 4.3.1 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 "true" and "false" for booleans instead of the properly capitalized versions ("True"

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 4.2: 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.

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. *Porta thus provides a convenient event- and time-based index into the underlying raw screencast videos that it records.*

Summary: Porta allowed test participants to give more specific and targeted feedback on how to improve tutorials.

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 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 4.3: 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.

4.3.2 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 4.3.1 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 [NKA01] 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.

All three explored Porta's positional heatmaps to see where students spent relatively more time and clicked on event markers to see what students were doing at those locations along with what errors they encountered. For instance, the Python tutorial's creator did not realize that Python syntax to demarcate code blocks with colon and whitespace was a major leap for novices. He only realized this 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 like 'less' and 'cat'. The Web Design tutorial's creator was surprised that students copy-pasted code with placeholders without filling them in with their own values; to him, it seemed obvious that, say, the value of the HTML 'href' attr could not literally be the "..." characters.

Ideas for improving tutorials: Instructors were also able to use Porta to come up with actionable ideas for improving their tutorials. The lower right of Table 4.3.1 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. He now plans to remove this section. The Web Design tutorial's creator concluded that he should have made CSS style changes more obviously salient. In the current tutorial, the visual changes were too subtle to notice.

During an end-of-session debriefing, all three instructors mentioned that if Porta were used to regularly test tutorials, it would serve as a forcing function for them to continuously improve their tutorials. The

All three instructors wanted to use Porta. The Python tutorial's creator said, "you're very rarely sitting down with students when building tutorials, so feedback is lacking." The Git tutorial's creator said that "the perennial problem is motivating MOOC creators to update their content because they don't know what to change. Going back and fixing things is painful without direct feedback and that they "rarely re-evaluated the value of existing tutorials.". Porta could help with that."

Finally, the Web Design tutorial's creator noted that "students very rarely ever give feedback about the state of existing materials" except when there are obvious bugs.

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.

Summary: Porta allowed tutorial creators to discover surprising insights about student behavior and also come up with specific actionable ideas for improving their tutorials.

4.4 Discussion: System Scope and Limitations

Figure 4.6 gives an overview of all the improvements over all 3 tutorials by both the learners and creators.

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; then Porta can integrate their events into its profiling visualization.

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. The experimenter can brief test participants on privacy implications and erase their test accounts after each session. When used in online experiments, participants must both be technically adept enough to install it and also put their trust in it. To allay concerns about privacy leaks, participants can inspect the raw data that Porta collects and view the profile visualizations, which is exactly what the tutorial creator sees. These logistical 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.

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.

4.5 Acknowledgment

The content from this chapter is borrowed from "Porta: Profiling Software Tutorials Using Operating-System-Wide Activity Tracing", Mysore et al. Under review at ACM UIST 2018

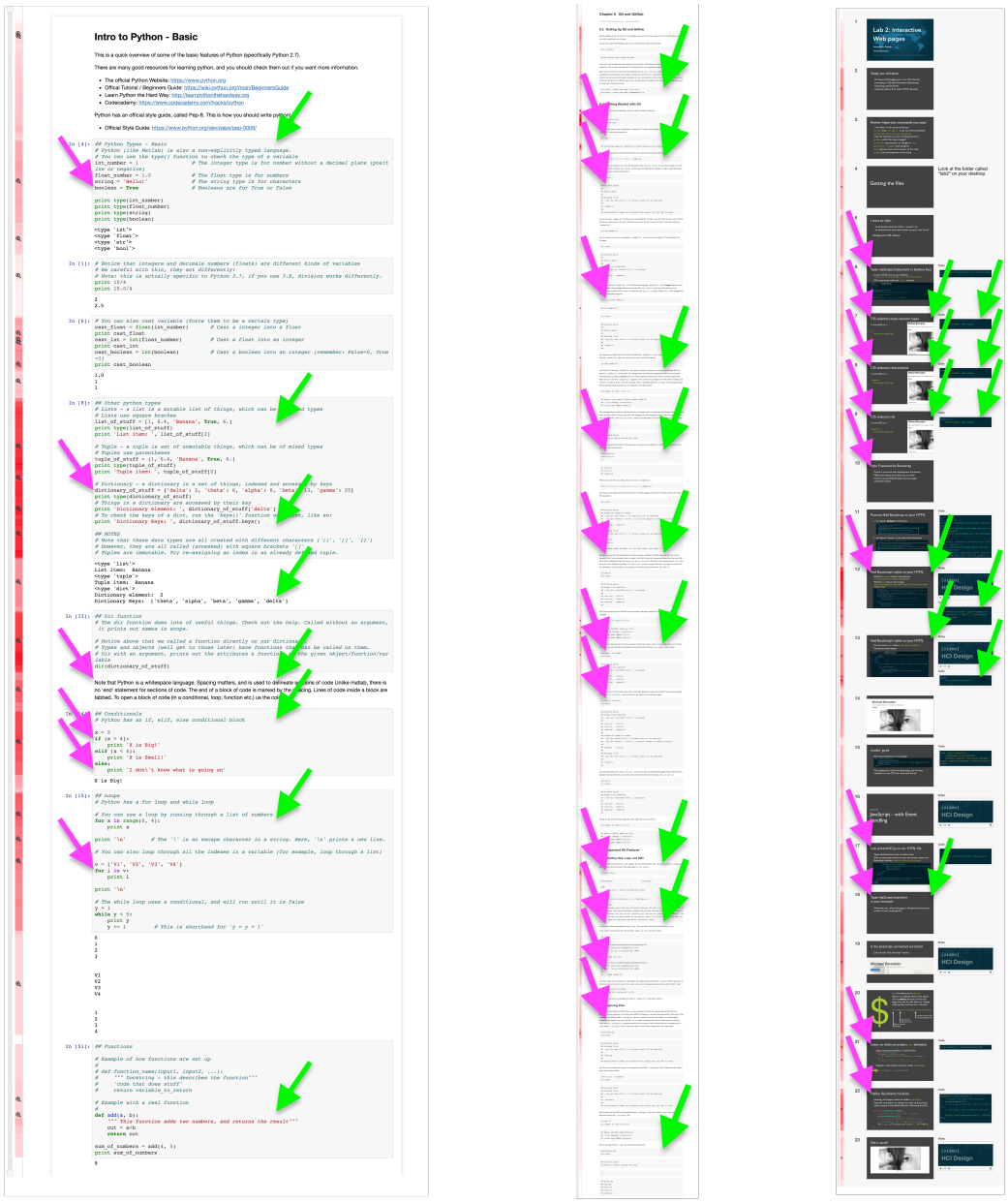

Figure 4.6: Screenshots of all 3 tutorials with green arrows pointing to the improvements identified by the learners and pink arrows pointing to the improvements identified by the creators

Chapter 5

Conclusion

To address the challenge of creating step-by-step tutorial with the same ease of creating a screencast, we created Torta, an end-to-end system for recording, editing, and consuming mixed-media tutorials that span multiple GUI and command-line applications. The core technical insight that underpins Torta is that the operating system already keeps track of vital filesystem and process-level metadata necessary for segmenting tutorial steps. Thus, combining operating-system-wide activity tracing with screencast video recording makes it possible to quickly create complex GUI and command-line app tutorials by demonstration. Torta's application-agnostic design makes it well-suited for creating tutorials in domains such as software development, data science, system administration, and computational research.

To address the challenges of providing effective feedback on the contents of software tutorials, we created Porta, a system that automatically builds *tutorial profiles* by tracking user activity within a tutorial webpage and across multiple applications. 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. We found via a user study of 3 tutorial creators and 12 learners that Porta helped both sets of users reflect more concretely about what parts

of the tutorials caused the most trouble and what could potentially be improved.

In conclusion, We believe that the systems described in this thesis make a very strong case toward the use of OS-wide monitoring for both creating new tutorials and profiling existing tutorials.

Torta opens up the possibilities of future tutorial systems that could make it much easier to create step-by-step tutorials through demonstration, this allowing for the proliferation of quality instructional material

Porta on the other hand could enable low cost user testing of instructional material and could lead to improvements of existing tutorials and documentation online.

Chapter 6

Future work

We built a modular OS monitoring and logging infrastructure on both the systems. As a result, we can easily incorporate additional OS level events on them. Future work for these systems could look at incorporating application level events into the logging system. An example of this is logging events on applications like Adobe Photoshop [Pho18] so that we can extend our systems to generate and profile tutorials that involve Photoshop.

In terms of user studies, we are currently working on a second user study for the Porta system. We want to mathematically model the number of learners a tutorial would need to reach a reasonable level of polish.

Also, future work includes a more in-depth consumer study on Torta. We want to look at the effectiveness of tutorials created by Torta. This includes a comparative study of the knowledge gain in users who used tutorials created by Torta and whether it was on par or better than static tutorials or screencasts.

Bibliography

[ACF+09] Jason Alexander, Andy Cockburn, Stephen Fitchett, Carl Gutwin, and Saul Greenberg. 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, pages 1665–1674, New York, NY, USA, 2009. ACM.

[Aut17] Autodesk screencast: A simple way to share what you know. `https://knowledge.autodesk.com/community/screencast`, 2017.

[BCLO05] Lawrence Bergman, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. Docwizards: A system for authoring follow-me documentation wizards. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, UIST '05, pages 191–200, New York, NY, USA, 2005. ACM.

[BDL+14] Andrea Bunt, Patrick Dubois, Ben Lafreniere, Michael A. Terry, and David T. Cormack. Taggedcomments: Promoting and integrating user comments in online application tutorials. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 4037–4046, New York, NY, USA, 2014. ACM.

[BWR+05] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, UIST '05, pages 163–172, New York, NY, USA, 2005. ACM.

[CAR+12] Pei-Yu Chi, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. Mixt: Automatic generation of step-by-step mixed media tutorials. In *Proceedings of the 25th Annual ACM Symposium on*

User Interface Software and Technology, UIST '12, pages 93–102, New York, NY, USA, 2012. ACM.

[CKW12] Parmit K. Chilana, Andrew J. Ko, and Jacob O. Wobbrock. Lemonaid: Selection-based crowdsourced contextual help for web applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1549–1558, New York, NY, USA, 2012. ACM.

[CLD14] Pei-Yu Chi, Bongshin Lee, and Steven M. Drucker. Demowiz: Reperforming software demonstrations for a live presentation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1581–1590, New York, NY, USA, 2014. ACM.

[CSL04] Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Leventhal. Dynamic instrumentation of production systems. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, Berkeley, CA, USA, 2004. USENIX Association.

[DHF+17] Biplab Deka, Zifeng Huang, Chad Franzen, Jeffrey Nichols, Yang Li, and Ranjitha Kumar. 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, pages 727–736, New York, NY, USA, 2017. ACM.

[DHK16] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. ERICA: Interaction mining mobile apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 767–776, New York, NY, USA, 2016. ACM.

[ffm17] Ffmpeg: A complete, cross-platform solution to record, convert and stream audio and video. `https://ffmpeg.org/`, 2017.

[FGF11] Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. Sketch-sketch revolution: An engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 373–382, New York, NY, USA, 2011. ACM.

[FLCT14] Adam Fourney, Ben Lafreniere, Parmit Chilana, and Michael Terry. Intertwine: Creating interapplication information scent to support coordinated use of software. In *Proceedings of the 27th Annual ACM Sym-*

posium on User Interface Software and Technology, UIST '14, pages 429–438, New York, NY, USA, 2014. ACM.

[Fox18] Paul D. Fox. Linux port of dtrace. `https://github.com/dtrace4linux/linux`, 2018.

[ful18] Fullstory: Pixel-perfect session replay. `https://www.fullstory.com/`, 2018.

[GAL+09] Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. Generating photo manipulation tutorials by demonstration. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 66:1–66:9, New York, NY, USA, 2009. ACM.

[GKM82] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. Gprof: A call graph execution profiler. In *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*, SIGPLAN '82, pages 120–126, New York, NY, USA, 1982. ACM.

[GM09] Max Goldman and Robert C. Miller. Codetrail: Connecting source code and web resources. *J. Vis. Lang. Comput.*, 20(4):223–235, August 2009.

[GMF10] Tovi Grossman, Justin Matejka, and George Fitzmaurice. Chronicle: Capture, exploration, and playback of document workflow histories. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 143–152, New York, NY, USA, 2010. ACM.

[GS12] Philip J. Guo and Margo Seltzer. Burrito: Wrapping your lab notebook in computational infrastructure. In *Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance*, TaPP'12, Berkeley, CA, USA, 2012. USENIX Association.

[Guo12] Philip J. Guo. *Software Tools to Facilitate Research Programming*. PhD thesis, Stanford University, May 2012.

[HDC11] Björn Hartmann, Mark Dhillon, and Matthew K. Chan. 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, pages 2207–2210, New York, NY, USA, 2011. ACM.

[HHWM92] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. Edit wear and read wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 3–9, New York, NY, USA, 1992. ACM.

[hot18] Hotjar: All-in-one analytics & feedback. `https://www.hotjar.com/`, 2018.

[HT07] Jeff Huang and Michael B. Twidale. Graphstract: Minimal graphical help for computers. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 203–212, New York, NY, USA, 2007. ACM.

[KGC+14] Juho Kim, Philip J. Guo, Carrie J. Cai, Shang-Wen (Daniel) Li, Krzysztof Z. Gajos, and Robert C. Miller. 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, pages 563–572, New York, NY, USA, 2014. ACM.

[KK10] Nate Kushman and Dina Katabi. Enabling configuration-independent automation by non-expert users. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 223–236, Berkeley, CA, USA, 2010. USENIX Association.

[Kle17] Scott Klemmer. Ucsd interaction design cogs120/cse170 - winter 2017. `http://ixd.ucsd.edu/home/w17/index.php`, 2017.

[Kro14] Rebecca P. Krosnick. Videodoc: Combining videos and lecture notes for a better learning experience. Master's thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 2014.

[LBM14] Tom Lieber, Joel R. Brandt, and Rob C. Miller. Addressing misconceptions about code with always-on programming visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2481–2490, New York, NY, USA, 2014. ACM.

[LBP+07] Oren Laadan, Ricardo A. Baratto, Dan B. Phung, Shaya Potter, and Jason Nieh. Dejaview: A personal virtual computer recorder. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 279–292, New York, NY, USA, 2007. ACM.

[LGF13] Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. Community enhanced tutorials: Improving tutorials with multiple demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1779–1788, New York, NY, USA, 2013. ACM.

[LGMF14] Ben Lafreniere, Tovi Grossman, Justin Matejka, and George Fitzmaurice. Investigating the feasibility of extracting tool demonstrations from in-situ video content. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 4007–4016, New York, NY, USA, 2014. ACM.

[LNL+10] Ian Li, Jeffrey Nichols, Tessa Lau, Clemens Drews, and Allen Cypher. Here's what i did: Sharing and reusing web activity with actionshot. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 723–732, New York, NY, USA, 2010. ACM.

[MDW+17] Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 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, pages 299–310, New York, NY, USA, 2017. ACM.

[MG17] Alok Mysore and Philip J. Guo. 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, pages 703–714, New York, NY, USA, 2017. ACM.

[MGF13] Justin Matejka, Tovi Grossman, and George Fitzmaurice. Patina: Dynamic heatmaps for visualizing application usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 3227–3236, New York, NY, USA, 2013. ACM.

[mim18] Mdn web docs: Mime types. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types`, 2018.

[mou18] Mouseflow - session replay, heatmaps, funnels, forms & user feedback. `https://mouseflow.com/`, 2018.

[NKA01] Mitchell J Nathan, Kenneth R Koedinger, and Martha W Alibali. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In *Proceedings of the third international conference on cognitive science*, pages 644–648. Beijing: University of Science and Technology of China Press, 2001.

[NL15] Cuong Nguyen and Feng Liu. Making software tutorial video responsive. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1565–1568, New York, NY, USA, 2015. ACM.

[PDL+11] Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. Pause-and-play: Automatically linking screencast video tutorials with applications. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 135–144, New York, NY, USA, 2011. ACM.

[Pho18] Autodesk photoshop: Edit videos, simulate real-life paintings, and more. `https://www.adobe.com/products/photoshop.html`, 2018.

[Pir07] Peter L. T. Pirolli. *Information Foraging Theory: Adaptive Interaction with Information*. Oxford University Press, Inc., New York, NY, USA, 1 edition, 2007.

[pos17] Developing apis is hard. postman makes it easy. `https://www.getpostman.com/`, 2017.

[PRHA14] Amy Pavel, Colorado Reed, Björn Hartmann, and Maneesh Agrawala. Video digests: A browsable, skimmable format for informational lecture videos. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 573–582, New York, NY, USA, 2014. ACM.

[Rus18] Mark Russinovich. Microsoft sysinternals: Process monitor v3.50. `https://docs.microsoft.com/en-us/sysinternals/downloads/procmon`, 2018.

[SE94] Amitabh Srivastava and Alan Eustace. 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, pages 196–205, New York, NY, USA, 1994. ACM.

[tob11] 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.

[WCC+14] Cheng-Yao Wang, Wei-Chen Chu, Hou-Ren Chen, Chun-Yen Hsu, and Mike Y. Chen. Evertutor: Automatically creating interactive guided tutorials on smartphones by user demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 4027–4036, New York, NY, USA, 2014. ACM.