Torta: Generating Mixed-Media GUI and Command-Line App Tutorials Using Operating-System-Wide Activity Tracing

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

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

ABSTRACT

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 GUI and command-line applications. To address this challenge, we created 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. The core technical insight that underpins Torta is that combining operating-system-wide activity tracing and screencast recording makes it easier to generate mixed-media (text+video) tutorials that span multiple GUI and command-line apps. An exploratory study on 10 computer science teaching assistants (TAs) found that they all preferred the experience and results of using Torta to record programming and sysadmin tutorials relevant to classes they teach rather than manually writing tutorials. A follow-up study on 6 students found that they all preferred following the Torta tutorials created by those TAs over the manually-written versions.

Author Keywords

tutorial generation; mixed-media tutorials; software tutorials

ACM Classification Keywords

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

INTRODUCTION

Step-by-step tutorials are vital for helping people such as software developers, data scientists, researchers, and system administrators perform complex software-based tasks. These tasks often require delicate coordination across multiple applications and depend on intricate operating-system-wide state. For example, if a programmer wants to start building a modern full-stack web app, they may need to first install Node.js and the npm package manager, run a slew of npm commands to configure a custom toolchain with a CSS preprocessor and a JavaScript code bundler, adjust OS environment variables to detect all required library dependencies and execution paths, customize their IDE to hook up to that toolchain, install web browser extensions for debugging, and set up a pipeline to deploy code to production servers. Since lots of independent command-line and GUI apps need to be separately configured and customized for each user, it is rare to find a "one-click installer" that automates this entire process end-to-end. Thus, many people often rely on online tutorials to figure out how to set up and debug all of the requisite steps.

Despite the importance of tutorials, we found through formative interviews that it is difficult to manually create step-by-step tutorials for complex tasks that span multiple GUI and command-line applications. 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 [20], and videos can be difficult to browse and navigate [17, 24]. Finally, regardless of format, it is hard for people to know whether they are following each step of the tutorial properly since tutorials are simply static content (e.g., text or video) that cannot give any personalized feedback.

To overcome these limitations, we introduce a new approach to making tutorials that: a) automatically records a screencast video along with OS-level events such as filesystem modifications and process invocations, b) generates *mixed-media tutorials* [7] with the benefits of both text and video formats, and c) gives step-by-step feedback to tutorial followers.

The core technical insight that underpins our approach 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 and screencast video recording makes it easier to generate tutorials that span multiple GUI and command-line applications. Our approach is novel due to its application-agnostic nature, since prior systems for creating mixed-media tutorials operate either wholly within a single application (e.g., an image editor [7, 11, 12, 21], IDE [4], or web browser [22]) or on a small fixed set of applications (e.g., Firefox+GIMP [10]).

UIST 2017, October 22–25, 2017, Quebec City, QC, Canada

ACM 978-1-4503-4981-9/17/10...$15.00
https://doi.org/10.1145/3126594.3126628

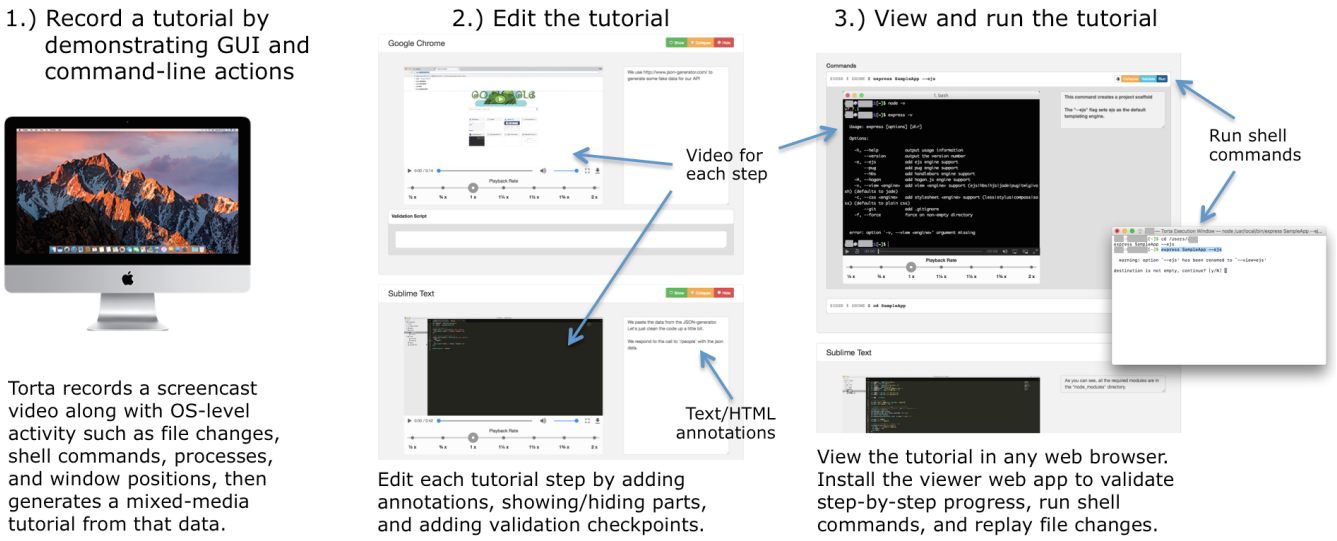

Figure 1. Torta allows macOS users to create mixed-media tutorials by demonstration, and then edit, view, and run those tutorials in a web browser.

We instantiated our approach in a prototype called Torta[1] for Apple's macOS (née "OS X"). Figure 1 shows an overview of how Torta can be used to both create and consume tutorials:

1. The tutorial creator first demonstrates the intended actions on their computer by running shell commands, launching GUI applications, and interacting with application windows. Torta automatically records a screencast video of their desktop along with a timestamped trace of OS-level activity that includes filesystem modifications, file versions, shell commands, window positions, and keystrokes.

   From this single demonstration, Torta automatically generates a mixed-media tutorial [7] that hierarchically segments the screencast video by foreground GUI windows, executed commands, and versions of saved files. It displays each segment as an individual step on a tutorial webpage.

2. However, since this initial demonstration likely contains redundancies or errors due to the difficulty of recording a pristine and error-free video demo in one take, Torta provides a novel user interface for editing tutorials prior to publishing. The tutorial editor uses data from both the recorded screencast video and OS-level activity traces to allow creators to compress and summarize portions of the tutorial, add textual annotations, insert file path templates that generalize the tutorial's contents across machines, and add checkpoints for viewers to validate their progress.

3. Torta-generated tutorials ("Tortorials") are ordinary webpages that mix text and embedded video, so people can consume them just like any web-based tutorial. Tortorials are also hierarchical, so users can zoom in to view more details on demand. If someone wants interactive feedback as they are following along, they can optionally install a Torta viewer app on their computer. Doing so enables them to use an augmented tutorial viewer that provides checkpoints to validate their progress at each step. The viewer app can also automatically run certain steps for the user.

[1]**T**ransparent **O**perating-system **R**ecording for **T**utorial **A**cquisition

Torta points toward a future where making complex software tutorials becomes as simple as interacting normally with the desired applications and adding some annotations afterward. It is best-suited for creating tutorials in operating-system-activity-heavy domains such as software development, data science, system administration, and computational research.

For tutorial creators, Torta aims to provide the best of both modalities – the fluid ease of demonstrating a set of computer actions *in-situ*, and the detailed rigor of writing text-based tutorials. For tutorial consumers, Torta allows them to browse hierarchically at the level of detail suitable for their needs and to get step-by-step feedback on their incremental progress.

To compare Torta with manually-written tutorials, we ran a pair of exploratory user studies. In the first study, we had 10 computer science teaching assistants (TAs) create software setup and programming tutorials relevant to the classes that they teach by both recording with Torta and by manually writing them in Google Docs. All 10 preferred using Torta due to its support for in-situ recording with think-aloud and to its automatic tracking of commands, file changes, and code-related diffs. They also all self-reported that their Torta-generated tutorials were better organized and higher quality. To get student feedback on those tutorials, we ran a follow-up study where we had 6 undergraduate students follow both the Torta and Google Docs versions of the tutorials created by TAs in the first study. All 6 preferred Torta's format and pointed to advantages such as better context, structure, and information-density, and the ability to validate step-by-step progress.

The contributions of this paper are:

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

RELATED WORK

Torta builds upon ideas in three prior lines of work: mixed-media tutorials, generating tutorials by demonstration, and operating-system-wide user activity tracing. To our knowledge, Torta is the first attempt to combine these ideas to produce a system that generates mixed-media tutorials from user demonstrations augmented with OS-wide activity tracing.

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. [7] 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 [24] and VideoDoc [17] 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.

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 [11]. Chronicle [12] 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 [1] 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. [21] generated tutorials by instrumenting Pixlr, an online image editing app, to collect usage logs alongside a screencast video. MixT [7] 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 [22] 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 [4] instruments the Eclipse IDE to let people generate tutorials for software development within that IDE. InterTwine [10] 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.

Operating-System-Wide Activity Tracing

Torta's approach of tracing activity at the operating-system level (rather than within specific applications) 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. [23] 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 [8] captures mouse and keyboard events to augment videos with a visual overlay that aids people in giving live presentations. EverTutor [26] tracks multitouch events on Android smartphones to help users create step-by-step app tutorials. Graphstract [15] tracks mouse and keyboard events and takes mini-screenshots of the areas around those events to generate minimalistic static tutorials. While these systems are lightweight and work across arbitrary desktop applications, they do not trace higher-level events (e.g., GUI window state, executed commands, filesystem modifications) necessary for making the kinds of mixed-media tutorials that Torta makes.

In terms of higher-level OS activity tracing, KarDo [18] 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 [19] 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 [14] traces GUI activity, shell commands, and filesystem modifications in a similar way as Torta, 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 Torta, they are meant for tasks such as automation, OS state replication, and scientific workflow management, and thus lack design affordances for automatic tutorial generation.

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 [16].

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.

THE DESIGN AND IMPLEMENTATION OF TORTA

Torta consists of three components: a tutorial recorder, editor, and viewer (Figure 1). We now describe each in turn.

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 [6] 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 [6] 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 2, a Torta-generated tutorial has a hierarchical structure that aims to follow the design guidelines of Chi et al. [7]:

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 [3] 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 files were added, deleted, renamed, and modified by processes associated with the foreground GUI window during that step (Figure 4). We chose to visualize

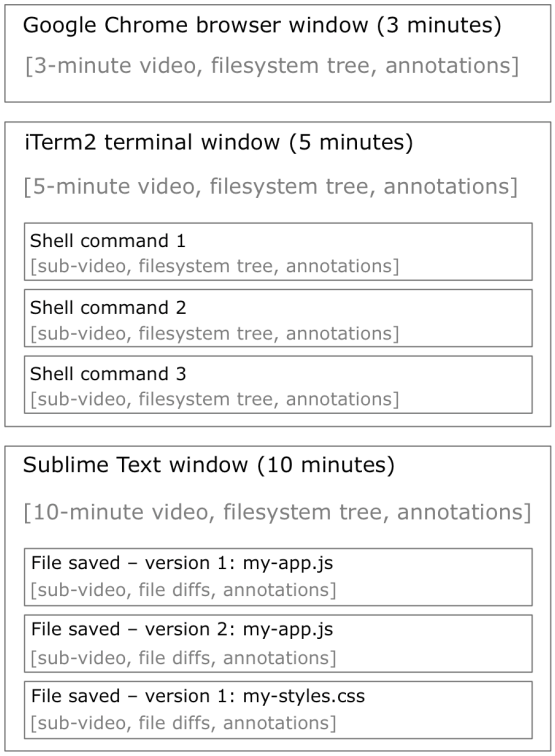

Figure 2. Example structure of a mixed-media tutorial generated by Torta. Each of the three steps represents a foreground GUI window duration. There are three sub-steps within the terminal and IDE windows.

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 2):

- ***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 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.

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

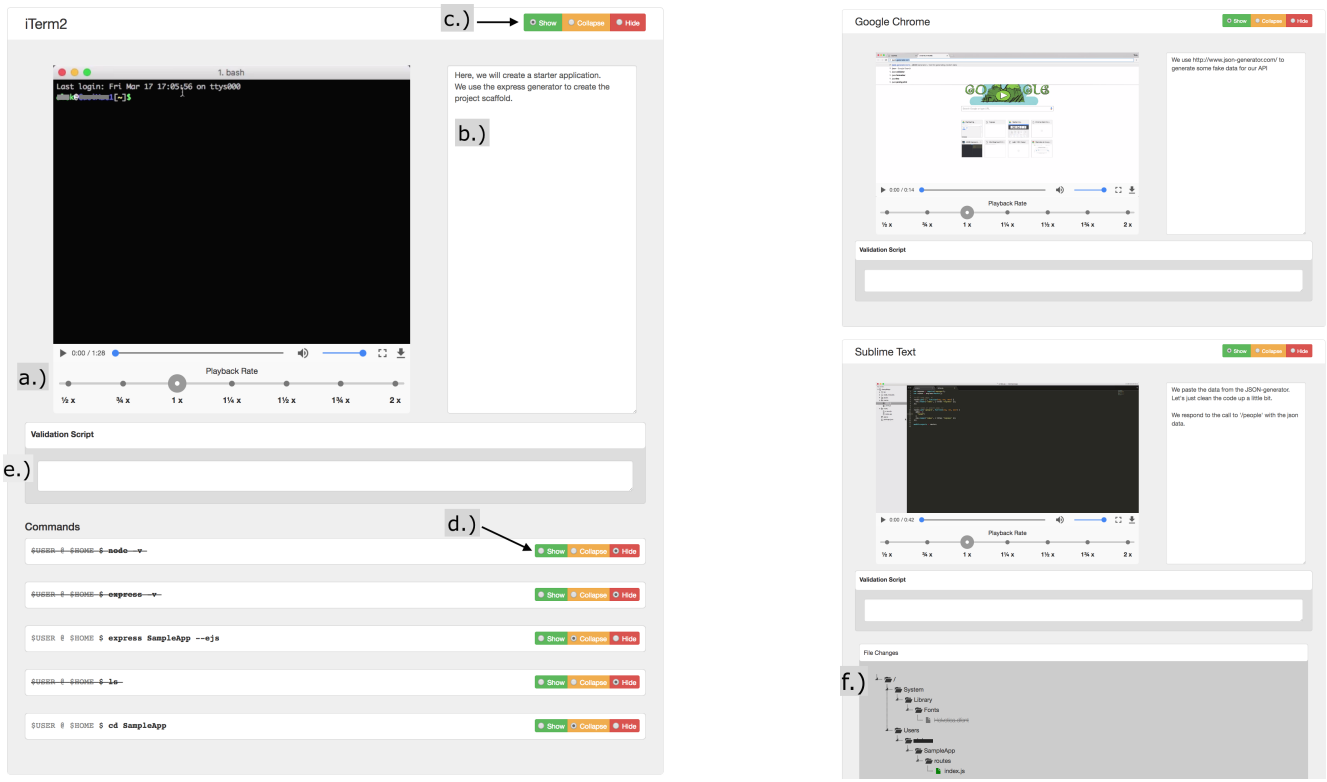

Figure 3. Zoomed-in screenshots of Torta's tutorial editor showing three steps (i.e., foreground windows): iTerm2, Google Chrome, and Sublime Text. Each step contains: a.) Video player with playback speed adjuster, b.) Text annotation box, c.) Toggle to show/collapse/hide this entire step, d.) Toggle 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 4).

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):

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.

Collapsing filesystem tree components: Recall that Torta displays a filesystem tree within each step and sub-step that modifies the filesystem (Figure 4). 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.

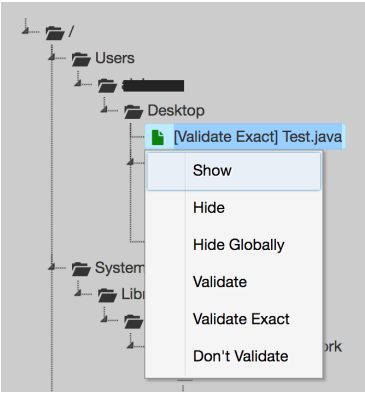

Figure 4. Each file-modifying tutorial step displays a filesystem tree of all files affected by running that step. In the editor UI, the user can right-click to show/hide files and to mark for validation.

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 4). 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 3e). 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 [9, 25].

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

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.

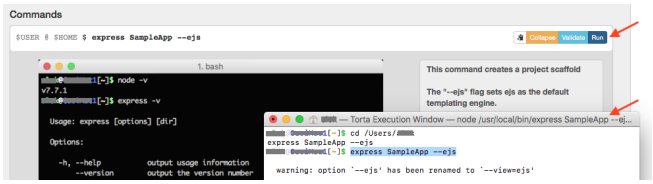

Figure 5. Each step and sub-step contains "Validate" and "Run" buttons on the upper right. Here when the user clicks on "Run" for a shell command sub-step, Torta runs that commands in a new terminal window.

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 Torta viewer web app on localhost, then they can access two additional features as shown in Figure 5:

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 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.

	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 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.

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.

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 1 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 1). 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 automatically.

- *Demonstrating GUI actions was much easier on video*: Subjects who heavily used GUI tools such as the Postman [2] 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 1 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.

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."*

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 6). 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 [14] 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

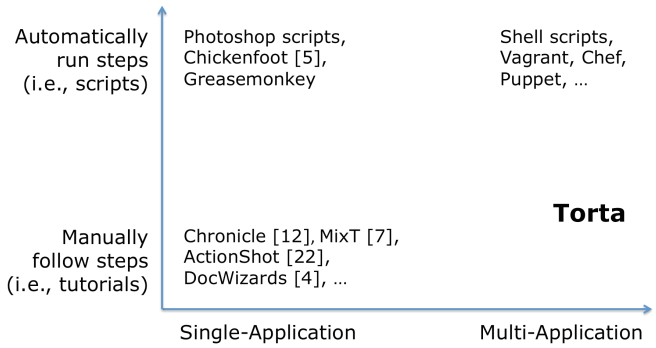

Figure 6. The design space of tools for generating step-by-step records of application actions, ranging from those that automatic run (i.e., scripts) to those meant for users to manually follow their steps (i.e., tutorials).

scripting languages, scientific libraries, and legacy research tools [13], so they can use Torta to generate documentation that can help colleagues reproduce and build upon their computational experiments. However, 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.

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.

CONCLUSION

We presented 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. We hope that it will inspire the design of future tutorial systems that bridge the gap between video- and text-based formats.

ACKNOWLEDGMENTS

Thanks to the UCSD Design Lab for feedback on early drafts and to the anonymous reviewers for their insightful feedback.

REFERENCES

1. 2017. Autodesk Screencast: A simple way to share what you know. https://knowledge.autodesk.com/community/screencast. (2017).
2. 2017. Developing APIs is hard. Postman makes it easy. https://www.getpostman.com/. (2017).
3. 2017. FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video. https://ffmpeg.org/. (2017).
4. Lawrence Bergman, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. 2005. 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)*. ACM, New York, NY, USA, 191–200. DOI: http://dx.doi.org/10.1145/1095034.1095067
5. Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. 2005. Automation and Customization of Rendered Web Pages. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*. ACM, New York, NY, USA, 163–172. DOI: http://dx.doi.org/10.1145/1095034.1095062
6. Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Leventhal. 2004. Dynamic Instrumentation of Production Systems. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '04)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=1247415.1247417
7. Pei-Yu Chi, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. 2012. 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)*. ACM, New York, NY, USA, 93–102. DOI: http://dx.doi.org/10.1145/2380116.2380130
8. Pei-Yu Chi, Bongshin Lee, and Steven M. Drucker. 2014. DemoWiz: Re-performing Software Demonstrations for a Live Presentation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1581–1590. DOI: http://dx.doi.org/10.1145/2556288.2557254
9. Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. 2011. 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)*. ACM, New York, NY, USA, 373–382. DOI: http://dx.doi.org/10.1145/2047196.2047245
10. Adam Fourney, Ben Lafreniere, Parmit Chilana, and Michael Terry. 2014. InterTwine: Creating Interapplication Information Scent to Support Coordinated Use of Software. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 429–438. DOI: http://dx.doi.org/10.1145/2642918.2647420
11. Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. 2009. Generating Photo Manipulation Tutorials by Demonstration. In *ACM SIGGRAPH 2009 Papers (SIGGRAPH '09)*. ACM, New York, NY, USA, Article 66, 9 pages. DOI: http://dx.doi.org/10.1145/1576246.1531372
12. Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2010. 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)*. ACM, New York, NY, USA, 143–152. DOI: http://dx.doi.org/10.1145/1866029.1866054
13. Philip J. Guo. 2012. *Software Tools to Facilitate Research Programming*. Ph.D. Dissertation. Stanford University.
14. Philip J. Guo and Margo Seltzer. 2012. BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure. In *Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance (TaPP'12)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=2342875.2342882
15. Jeff Huang and Michael B. Twidale. 2007. Graphstract: Minimal Graphical Help for Computers. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 203–212. DOI: http://dx.doi.org/10.1145/1294211.1294248
16. Scott Klemmer. 2017. UCSD Interaction Design COGS120/CSE170 - Winter 2017. http://ixd.ucsd.edu/home/w17/index.php. (2017).
17. Rebecca P. Krosnick. 2014. *VideoDoc: Combining Videos and Lecture Notes for a Better Learning Experience*. Master's thesis. MIT Department of Electrical Engineering and Computer Science, Cambridge, MA.
18. Nate Kushman and Dina Katabi. 2010. Enabling Configuration-independent Automation by Non-expert Users. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 223–236. http://dl.acm.org/citation.cfm?id=1924943.1924959

19. Oren Laadan, Ricardo A. Baratto, Dan B. Phung, Shaya Potter, and Jason Nieh. 2007. DejaView: A Personal Virtual Computer Recorder. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)*. ACM, New York, NY, USA, 279–292. DOI: http://dx.doi.org/10.1145/1294261.1294289

20. Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. 2013. Community Enhanced Tutorials: Improving Tutorials with Multiple Demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1779–1788. DOI: http://dx.doi.org/10.1145/2470654.2466235

21. Ben Lafreniere, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2014. 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)*. ACM, New York, NY, USA, 4007–4016. DOI: http://dx.doi.org/10.1145/2556288.2557142

22. Ian Li, Jeffrey Nichols, Tessa Lau, Clemens Drews, and Allen Cypher. 2010. 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)*. ACM, New York, NY, USA, 723–732. DOI: http://dx.doi.org/10.1145/1753326.1753432

23. Cuong Nguyen and Feng Liu. 2015. Making Software Tutorial Video Responsive. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 1565–1568. DOI: http://dx.doi.org/10.1145/2702123.2702209

24. Amy Pavel, Colorado Reed, Björn Hartmann, and Maneesh Agrawala. 2014. 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)*. ACM, New York, NY, USA, 573–582. DOI: http://dx.doi.org/10.1145/2642918.2647400

25. Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. 2011. 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)*. ACM, New York, NY, USA, 135–144. DOI: http://dx.doi.org/10.1145/2047196.2047213

26. Cheng-Yao Wang, Wei-Chen Chu, Hou-Ren Chen, Chun-Yen Hsu, and Mike Y. Chen. 2014. 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)*. ACM, New York, NY, USA, 4027–4036. DOI: http://dx.doi.org/10.1145/2556288.2557407