2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

Do Developers Discover New Tools On The Toilet?

Emerson Murphy-Hill, *Google, LLC*, emersonm@google.com
Edward K. Smith[*], *Bloomberg*, tedks@riseup.net
Caitlin Sadowski, *Google, LLC*, supertri@google.com
Ciera Jaspan, *Google, LLC*, ciera@google.com
Collin Winter[*], *Waymo*, collinwinter@waymo.com
Matthew Jorde, *Google, LLC*, majorde@google.com
Andrea Knight, *Google, LLC*, aknight@google.com
Andrew Trenk, *Google, LLC*, atrenk@google.com
Steve Gross, *Google, LLC*, stevegross@google.com

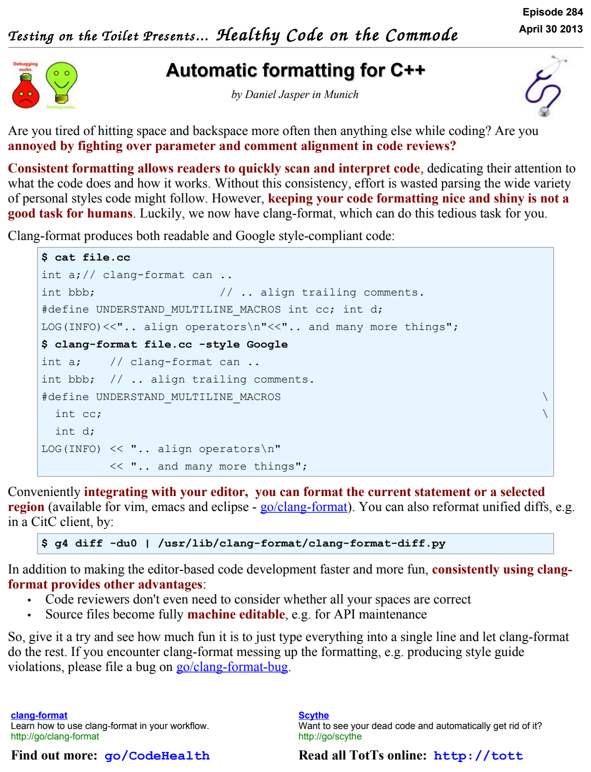
***Abstract*—Maintaining awareness of useful tools is a substantial challenge for developers. Physical newsletters are a simple technique to inform developers about tools. In this paper, we evaluate such a technique, called Testing on the Toilet, by performing a mixed-methods case study. We first quantitatively evaluate how effective this technique is by applying statistical causal inference over six years of data about tools used by thousands of developers. We then qualitatively contextualize these results by interviewing and surveying 382 developers, from authors to editors to readers. We found that the technique was generally effective at increasing software development tool use, although the increase varied depending on factors such as the breadth of applicability of the tool, the extent to which the tool has reached saturation, and the memorability of the tool name.**

I. Introduction

Tools can help increase developer productivity by increasing velocity and code quality. For instance, tools can find concurrency bugs [28], reduce the effort to analyze customer feedback [14], and help configure caching frameworks [10]. With an increasing number of tools becoming available for developers to use, the opportunity to improve productivity by increasing tool usage is enormous.

However, as the number of tools increases, so does the difficulty for developers to gain awareness of relevant tools. As Campbell and Miller argue, tools in major development environments suffer from "deep discoverability" problems [9]. The problem extends beyond software development; in Grossman and colleagues' survey of AutoCAD users, a "typical problem was that users were not aware of a specific tool or operation which was available for use" [20]. The problem is compounded at large companies like Microsoft [39], where developers create in-house tools and wish to share them with peers.

To increase awareness and adoption of software tools and practices, Google uses a technique called "Testing on the Toilet", or TotT for short (Figure 1). The TotT episodes are 1-page printed newsletters, written by developers and posted in restrooms [6]. While originally aimed at promoting testing tools and practices – hence the name – over the years TotT has become more inclusive of other kinds of software development practices and tools. Throughout the period of our study, episodes were distributed by volunteers; more recently, facilities staff have taken up distribution. Episodes are posted in restrooms for about a week, until the next episode is posted.



Episode 284
April 30 2013

Testing on the Toilet Presents... Healthy Code on the Commode

Debugging sucks.
Testing rocks.

Automatic formatting for C++

by Daniel Jasper in Munich

Are you tired of hitting space and backspace more often then anything else while coding? Are you **annoyed by fighting over parameter and comment alignment in code reviews?**

Consistent formatting allows readers to quickly scan and interpret code, dedicating their attention to what the code does and how it works. Without this consistency, effort is wasted parsing the wide variety of personal styles code might follow. However, **keeping your code formatting nice and shiny is not a good task for humans**. Luckily, we now have clang-format, which can do this tedious task for you.

Clang-format produces both readable and Google style-compliant code:

```
$ cat file.cc
int a;// clang-format can ..
int bbb;                  // .. align trailing comments.
#define UNDERSTAND_MULTILINE_MACROS int cc; int d;
LOG(INFO)<<".. align operators\n"<<".. and many more things";
$ clang-format file.cc -style Google
int a;    // clang-format can ..
int bbb;  // .. align trailing comments.
#define UNDERSTAND_MULTILINE_MACROS                                 \
  int cc;                                                           \
  int d;
LOG(INFO) << ".. align operators\n"
          << ".. and many more things";
```

Conveniently **integrating with your editor, you can format the current statement or a selected region** (available for vim, emacs and eclipse - go/clang-format). You can also reformat unified diffs, e.g. in a CitC client, by:

```
$ g4 diff -du0 | /usr/lib/clang-format/clang-format-diff.py
```

In addition to making the editor-based code development faster and more fun, **consistently using clang-format provides other advantages**:

- Code reviewers don't even need to consider whether all your spaces are correct
- Source files become fully **machine editable**, e.g. for API maintenance

So, give it a try and see how much fun it is to just type everything into a single line and let clang-format do the rest. If you encounter clang-format messing up the formatting, e.g. producing style guide violations, please file a bug on go/clang-format-bug.

clang-format
Learn how to use clang-format in your workflow.
http://go/clang-format

Find out more: `go/CodeHealth`

Scythe
Want to see your dead code and automatically get rid of it?
http://go/scythe

Read all TotTs online: `http://tott`

Fig. 1: TotT episode promoting CLANG-FORMAT.

Software developers have posted episodes at Google since May 2006, and other organizations have invested in similar efforts. One such example is the Schibsted Group's Testing on the Toilet, which uses a format very similar to our own [5]. Similarly, both Johns Hopkins Univer-

[*]Research performed while at Google.

sity and Harvard Law School publish restroom newsletters that occasionally contain tips about features of university-relevant software [2], [3]. To our knowledge, none of these efforts have been evaluated.

The main contribution of this paper is the first evaluation of the effectiveness of TotT for distributing knowledge about software development tools. While the approach has garnered media attention [7], [19], [11], little is known about TotT's overall effectiveness. To this end, we performed a case study, a research methodology appropriate for investigating "a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident" [43]. Our study analyzed the usage of 12 command-line development tools before and after the publication of a corresponding episode, to evaluate the following hypothesis:

> **Hypothesis**: Testing on the Toilet increases usage of advertised developer tools.

We apply CausalImpact, a Bayesian statistical technique that was developed to evaluate the impact of advertising on website traffic [8], to determine whether TotT had a statistically significant impact on tool usage. We then provide context for each tool's usage over time by interviewing and surveying software developers who read, edit, and author episodes. Our results suggest that the technique is generally effective at increasing tool awareness, although this increase varies depending on each software development tool and its context, such as the breadth of applicability of the tool, the extent to which the tool has reached saturation, and the memorability of the tool name.

II. Related Work

Studies of Tool Adoption. Diffusion of Innovations seeks to understand how people adopt new ideas [37]. In our study, we measure only the first phases of the diffusion of innovations, from knowledge to implementation, where developers gain awareness of the tool and begin to employ the tool. We assume that while developers may gain awareness and some knowledge about the tool through TotT, their decisions about whether to fully adopt a tool are driven by properties of the tool and the developers' work context, rather than by TotT. Thus, we do not expect that TotT itself drives long-term tool adoption, beyond inspiring developers to try the tool.

Several researchers have investigated diffusion of innovations in the field of software development. In earlier work using surveys, Fichman and Kemerer found that for database management systems, programming languages, and computer aided software engineering tools that are purchased, use of purchasing data is a poor predictor of actual adoption with companies [15]. Iivari's surveys found that lack of management support is a significant contributor for non-adoption of software-engineering tools [22]. More recently, Witschey and colleagues surveyed developers about their adoption of security tools; they found that the strongest predictor of a developer's likelihood to use a tool was their ability to observe their peers using tools [42], [41]. Murphy-Hill and colleagues interviewed software developers about how they first discovered tools, finding that the most effective way developers learned about new tools was through their peers [34]. In contrast to these studies, we rely on tool usage logs and statistical analysis to understand tool usage, which avoids the pitfalls of relying exclusively on self-reported data.

Studies of Restroom Advertising. Outside of software engineering, several previous research studies have evaluated restroom advertising. Hoffman found that 48 men could more often recall fliers placed above restroom urinals than in a study area [21]. Kaltenbaugh and colleagues found that 217 survey respondents reported that sports advertisements placed above restroom paper towel dispensers were easy to read and placed in an appropriate location [24]. Lehmann and Shemwell found that the majority of 146 survey respondents recalled features of an advertisement placed above urinals and near mirrors in restroom bars, even though respondents "might be somewhat impaired due to alcohol consumption" [26]. Our study goes beyond such self-reported perception and recall data collected in these prior studies, and instead relies on actual usage of the product.

Like our study, Mackert and colleagues evaluated the effect of advertisements on objective outcomes. In that study, researchers concealed for 30 hours in bathroom stalls observed the handwashing behavior of 1005 people, before and after introducing pro-handwashing lobby and restroom posters [30]. The posters did not significantly increase handwashing. In contrast, we studied several software engineering posters introduced at different points in time, and did not use covert restroom surveillance.

Approaches to Improving Tool Usage. Singer found that gamification can increase use of features of the git version control system [38]. Cockburn and Williams found that pair programming can influence the tools that developers discover [12].

Some systems recommend new tools by observing current tool use [27], [31], [32]. In the context of software development tools, Murphy-Hill and colleagues used collaborative filtering and other techniques to automatically recommend IDE commands [33]. Other recommender systems for software engineering have taken a task-oriented approach. Viriyakattiyaporn and Murphy's Spyglass system used patterns of tool non-usage to recommend program navigation tools [40]. Similarly, WitchDoctor and BeneFactor can recognize when a developer is refactoring without a tool, and then recommend completing the refactoring using the tool [16], [17].

While studies of these systems evaluate personalized recommender systems, ours evaluates a substantially different approach, that of non-personalized newsletters. Our study also investigates tool usage with an order of magnitude more developers than in those studies.

III. Methodology

To investigate Testing on the Toilet's effectiveness for promoting software development tools, we synthesized data from both quantitative and qualitative sources. We first analyzed the logs of command-line software development tools (Section III-A) before and after twelve episodes appeared (Section III-B). We then contextualized these results with qualitative data from three perspectives, that of episode authors (Section III-C), episode editors (Section III-D), and episode readers (Section III-E). Blank surveys and interview scripts are available online [1] to increase the replicability of our study.

A. Analysis of Tool Logs

Google collects log data of how employee software developers use command-line tools. Most development at Google occurs on Linux workstations using a uniform, centrally built toolchain; every binary built for the workstation fleet creates a syslog entry on start. This data provides the number of developers per day using each tool.

We use a statistical package called CausalImpact, which analyzes time series data to determine whether an intervention had an impact on that data [8]. For our analysis, we used daily unique users as the time series and the date on which the episode was released as the intervention. CausalImpact creates a model of the counterfactual of tool usage; that is, the expected usage of the tool had the intervention not occurred. We used three sources of data for the model:

- First, we used the usage of the tool before the episode was published. Intuitively, for example, if a tool gains one user per week before the episode is published, one would naturally expect the tool to continue gaining one user per week. CausalImpact then uses this data as an empirical hyperparameter to its Bayesian model.
- A second empirical hyperparameter is a set of control groups for which the intervention did not occur. Here we used other tools that were not promoted in an episode, that is, every other command-line tool used at Google, more than 10,000 tools in total. Although many of these tools bear little resemblance to our TotT tools, CausalImpact accounts for this by reducing the weights of dissimilar tools, effectively discarding them.
- Third, because usage across days is not uniform, especially on weekends, we included a 7-day seasonal hyperparameter into CausalImpact's model.

For each tool, CausalImpact needs a pre-period from which to measure trends in baseline tool usage, and a post-period against which to compare counterfactual tool use. To help establish these periods, we re-ran a sensitivity analysis by generating 972 CausalImpact models with pre- and post-periods of between one week and more than two years, using a dozen control tools to enable sufficiently fast analysis. The results were largely consistent across pre-periods of differing lengths; consequently, for the main analysis reported in this paper, we chose six months to balance being long enough to reasonably establish baseline usage, but short enough to exclude potentially confounding factors, such as other interventions like social media posts about the tool. For varying post-period lengths, tool usage rates were substantially different. Inspecting the data, there were two reasons: tools whose usage showed an initial uptick that later subsided, and tools whose usage is influenced by some later intervention. Consequently, we need to intelligently choose a post-period duration. Since we expect the effect of TotT itself to last only a short amount of time (Section II), we chose 3 three weeks as a reasonable period because each episode is scheduled to be posted for one week, but may linger for some time afterwards.

B. Episodes

We next gathered all episodes describing command-line software development tools for which we could gather sufficient data to draw conclusions. To achieve this goal, we used the following criteria:

- The episode must have been published during the six the years for which we had tool log data available. 255 episodes met this criterion.
- The episode must introduce a development tool that can be invoked on the command line so that we could capture tool log data. 27 episodes met this criterion.
- The episode must not have been published during a gap in the tool log data. Such gaps occurred three times, lasting between several weeks and several months, and affected all tool data. One tool had only six weeks of data between the start of the tool logs and publication of its episode, so we excluded this tool as well. A total of seven episodes were excluded using this criterion.
- The tool must have had at least 10 daily users at some point during the pre- or post-period. We excluded such tools because we found their data too sparse to reason about, either statistically or intuitively. Six episodes were excluded using this criterion.
- The episode author still worked at Google and agreed to be interviewed, to provide context. Two episodes were excluded using this criterion.

Twelve suitable episodes remained after this process.

C. Author Perspectives

To understand the context of usage for each software development tool, we characterized the process from the perspective of developers familiar with each tool: the episode authors. These authors are developers who write the content of an episode and refine it with feedback from domain subject matter experts, editors, and any other interested developers. In many but not all cases, these were developers who were involved with creating the tool. We additionally talked to one developer who authored four episodes that, coincidentally, were all published during the tool log gaps; while we don't discuss his episodes specifically, we include his interview data. We first asked interviewees about their expectations before publishing the episode and their recollection of whether it was successful. We then showed them a plot of their tool's usage before and after the episode, including CausalImpact's counterfactual predictions; we then asked them for explanations of trends in the tool's usage. The first author analyzed this data by transcribing audio interviews (one author declined audio recording; here we used handwritten notes), then open coded the transcripts. A secondary source of data we use were surveys of 59 episode authors, which asked respondents for feedback about the publishing process, and included questions about process satisfaction and how the developers measured the impact of their episode.

D. Editor Perspectives

To understand editors' perspectives on the effectiveness of TotT, we used two sources of data. First, we used documentation about the tools and the history of the TotT initiative. Second, we interviewed four developers involved in editing episodes. These editors are responsible for choosing which episodes are appropriate for publication, providing feedback on episodes, soliciting feedback from subject matter experts, and managing the publication process. We used the same qualitative methodology as we used with author interviews. We selected two developers who were involved in TotT during its inception, and two developers who are currently involved in TotT editing. We asked editors about their general expectations about the effect of publishing episodes about tools. We then named the tools we analyzed, allowed them to select a tool or two of interest, showed them the usage plots for those tools, and asked whether the plots matched their expectations.

E. Reader Perspectives

To understand the TotT reader perspective, we turned to several sources of data. To determine how many engineers actually encounter episodes, we surveyed developers on a general mailing list about their office location and viewing habits; we received 234 responses.

Because episode authors and editors may provide an especially positive view on the effectiveness of TotT, we specifically sought out countervailing viewpoints from authors in two ways. First, we analyzed internal communications on a channel that is known Google-wide for jokes about developer frustrations, searching for internal jokes that contained "TotT". Second, to determine why developers *do not* learn about a tool from TotT, we chose a tool with substantial usage after the episode, then surveyed developers who (a) became employees at least 2 months before the episode was published, (b) used the tool, but only started to do so 6 months or more after the episode was published, and (c) have invoked the tool at least 100 times. We inferred that these developers found the tool useful, then asked them why they didn't start using the tool immediately after the episode was published. 77 developers responded out of 276 surveyed.

F. Threats to Validity

Before we discuss the results of our study, it's worth discussing the major issues regarding validity of our methods.

- We assume that the number of unique users of a tool per day is good measure of software development tool discovery, but other measures of discovery could be used, such as developer self-reports.
- How well TotT would work at another company depends on the extent to which Google is similar to that company. Google is large, multi-site, software-focused, founded about 20 years ago, and based in Silicon Valley. Google uses a substantial amount of shared infrastructure used by most developers in the form of a monolithic repository [23].
- We focused on command-line tools, but tools invoked differently may experience distinct usage patterns when communicated through TotT.
- A post-TotT tool usage increase might not be caused by the episode. For example, the effects of co-occurring promotions of a tool will be conflated with the effects of the episode. Because CausalImpact has no way to control for this effect, we attempt to mitigate this by asking interviewees about other tool advertisements and pointing them out to the reader in the relevant plots.
- Authors were asked about episodes that were up to six years old; authors may have difficulty remembering the events surrounding the episode. To mitigate this, we brought the relevant episode for the author's reference.
- Tool usage may be influenced by fluctuations in distribution, which may vary from week to week. For example, volunteers who post episodes in the bathroom may go on vacation in a given week. Likewise, the quality of the writing in the episode may vary from episode to episode, but we did not attempt to measure quality.
- Our analysis of jokes is opportunistic but incomplete, because some benefits or drawbacks may not be amenable to jokes. Thus, the joke analysis cannot be expected to identify all benefits and drawbacks of TotT.

We used *member checking* [13] to improve validity overall by inviting editors and authors to comment on drafts of this paper. Fourteen interviewees provided comments on our results, ranging from minor wording to reinterpretation of existing plots. We adjusted our wording and interpretation, sometimes after a back-and-forth discussion.

IV. Tools Studied

We next provide brief descriptions of the 12 tools we studied. With the exception of CLANG-FORMAT and IBLAZE, we have renamed the tools in this paper for confidentiality reasons.

a) Code Formatters.: **clang-format** is an open source code formatter for C, C++, Objective-C, JavaScript, Java, and protocol buffers [4]. Google has a strictly-enforced coding style for each of these languages. **PythonFormatter** is similar to CLANG-FORMAT, but is targeted at Python. PythonFormatter is an internal tool that removes Python lint errors and reformats the source code.

b) Build and Integration Tools.: **iblaze** is a wrapper around blaze, our centralized build tool; blaze is used by most developers at Google. iblaze re-runs blaze whenever a dependency of any target affected by the command is modified. This allows a developer to see build or test results without having to manually re-run a command. **VerifyDetermBuild** is a tool designed to debug long builds. The build system reuses cached build results when possible, but if binaries change between builds (e.g. if a timestamp is included in the binary), the new binary causes a cache miss. The tool VerifyDetermBuild runs two builds, then reports any outputs that differ. The developer then diagnoses and fixes the problem. **EmulatorSettings** is a wrapper script on top of the standard emulator for the Android mobile operating system, using default settings designed for interactive development and debugging, optimized for our internal development environment.

c) Quality Assurance Tools.: **SandboxManager** is a framework for describing, launching, and tearing down full system stacks, or sandboxes, for end-to-end and exploratory testing. **UIDiff** is a tool used to evaluate changes to web application user interfaces. Users are able to take screenshots showing how their application appears in different web browsers, both before and after a proposed change; the resulting images are shown side-by-side together with a visual diff. **Coverage** is a test coverage tool (the real name of this tool is a German word that may be difficult to remember for those who do not speak German). While Coverage is typically run automatically, here we analyze the manual invocations, which are useful when the a developer wants to try the tool or check code coverage before sending a change for review.

d) Production Tools.: **EstimateResources** analyzes changes to the configuration of production server jobs, to see if the job will still fit into the resources (e.g. memory) available on the machines and clusters on which they run. **ChangeTimeZone** can assist developers who go "on call", so that they can quickly respond to problems in deployed software. Developers specify on-call periods in those files using the time zone of Google's headquarters, for historical reasons. Consequently, developers not in that time zone must do the conversion manually. ChangeTimeZone automates this conversion by letting the developer specify times in any time zone.

e) Other Tools.: **GenerateDoc** is an documentation generator for source code packages. It uses the number of imports to rank source files by their popularity, and generates a README file in Markdown that includes the most popular source files, the number of references to them, and text from their top-level documentation comments. **PatchSearch** searches commits, where developers can issue queries like "What commits have I merged in the last year?" Developers can issue queries through a web interface or through the the command-line tool that we analyze here. While the web interface is more commonly used, the command line version is especially useful as part of scripts.

	Absolute	**Relative**	**p**
SandboxManager	-2.4	-15%	.052
UIDiff	1.2	26%	.017
EstimateResources	8.3	58%	.001
Coverage	9.1	68%	.001
EmulatorSettings	15	80%	.001
ChangeTimeZone	2.7	111%	.018
PatchSearch	14	152%	.007
clang-format	138	189%	.002
iblaze	264	378%	.001
VerifyDetermBuild	12	1000%	.001
PythonFormatter	63	1710%	.001
GenerateDoc	28	5233%	.001

TABLE I: The effect of TotT on tool usage. Absolute means the number of unique daily users over the counterfactual. Relative means the percent increase of daily users over the counterfactual.

V. Results

The core of our results explore quantiatively whether TotT is effective (Section V-A). We then describe sitautions when TotT is not effective (Section V-B), explore factors beyond TotT that influence tool usage (Section V-C), compare TotT to other tool knowledge distribution techniques (Section V-D), and explain outcomes of TotT beyond spurring tool usage (Section V-E).

A. TotT's Effectiveness

Table I quantitatively summarizes the results of the 12 tools we studied. The first column indicates the name of the tool. The second column indicates the absolute effect, the total number of unique daily users gained over the counterfactual predicted by CausalImpact. The third column indicates the relative effect, the percent increase of daily users over the counterfactual. The final column indicates the p-value, the counterfactual probability of a causal effect. The rows are ordered by relative effect. To take an example from Table I, iblaze had about 264 more users per day, compared to the predicted value, over the three week period after its episode. This 378% increase was statistically significant.

Figure 2 shows the usage of each of the tools before and after each tool's episode. For example, Figure 2a shows an example plot of the usage of the tool clang-format before and after it was promoted in TotT. In each plot, the x-axis represents time and the y-axis represents the number of unique developers using that tool on that day. The solid black line represents the actual usage of the tool; all tools exhibit a weekly cycle, in which each tool is used less on the weekends. Translucent bubbles represent first-time users of a tool; the larger the bubble, the more first time users that day. Vertical lines indicate dates of interest: a solid grey line indicates the day prior to the tool's episode being published; the grey dotted line afterwards represents

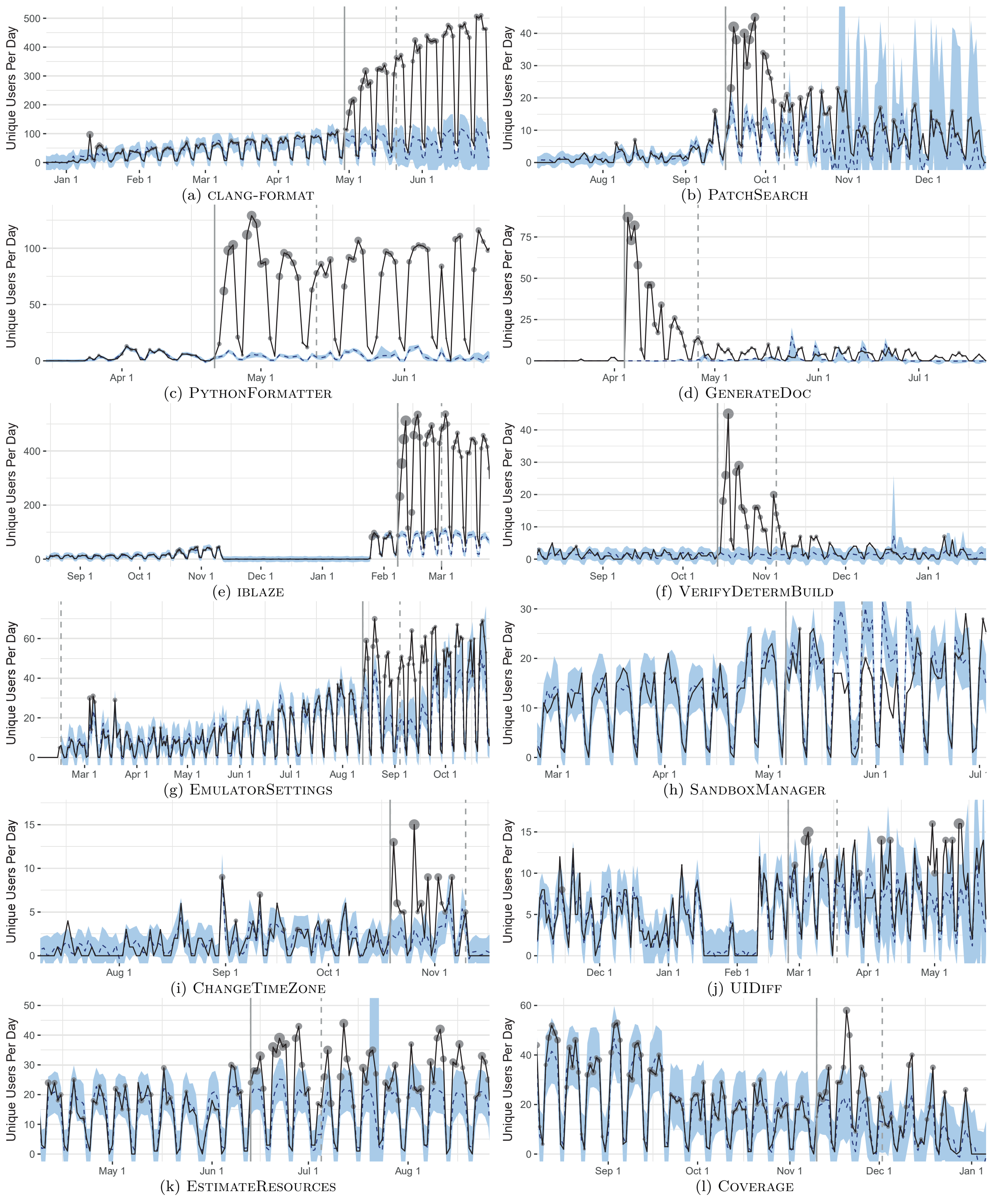


Fig. 2: Daily tool usage rates, before and after episodes (solid grey vertical lines).

3 weeks after the episode was published; and, shown in some plots, the grey dotted line prior represents the six months before the episode was published. This last line is shown only when the corresponding date is visible on the plot. The blue dotted line represents the counterfactual usage, predicted by CausalImpact. The blue shaded region around the blue line represents the 95% confidence interval for the prediction. As explained in Section III-B, gaps in the underlying usage data are visible for some tools, most obviously for IBLAZE (Figure 2e, mid-November through late January). In each plot we show different x-axis limits to illustrate particular features of each time series, but online we include plots over full 1-year periods [1].

Overall, the results suggest that usage of all tools but one were increased as a result of their promotion via TotT, confirming our hypothesis that TotT increases usage of advertised developer tools. However, on visual inspection of Figure 2, several usage patterns emerge. In the top plots (Figures 2a– 2f), we see strong growth after the episode is released. In the case of the three plots on the top-left quadrant (2a, 2c, 2e), growth is sustained. In contrast, the plots in the top-right quadrant (2b, 2d, 2f) show a drop-off after the episode is released. This contrast may be attributed to each tools' designs; CLANG-FORMAT, PYTHONFORMATTER, and IBLAZE are all designed to be used on a daily basis, while PATCHSEARCH, GENERATEDOC, and VERIFYDETERMBUILD are designed to be used on occasion. The bottom plots (Figures 2g–2l) exhibit weaker growth.

B. What Limits TotT's Effectiveness

Although TotT appears effective at distributing tool knowledge, our results suggest several challenges.

1) Irregular Coverage: Interviewees complained about uneven coverage of episodes amongst offices and buildings. When we surveyed developers via internal mailing lists about this, several smaller offices confirmed uneven coverage; apparently some offices did not have volunteers distributing episodes. In our distributed offices that received more than 10 responses each (one in the US and two in Europe), 78% of respondents reported usually seeing an episode in their restrooms. At our headquarters, which contains dozens of buildings, only 47% of respondents reported usually seeing an episode in their restroom. This finding was confirmed by our other set of survey respondents, the developers who use IBLAZE now but did not around the time the episode appeared. Of those, 21 respondents were certain that they had not seen the episode, often blaming poor coverage in their building. The editors confirmed that this is sometimes the case, as TotT is distributed by volunteer developers; when those developers go on vacation, move around, or leave the company, distribution can be disrupted. Since facilities staff have taken up distribution recently, this problem should be alleviated.

Similarly, developers made jokes that expressed frustration about episodes in restrooms not being regularly updated. On the other hand, slow-to-update episodes can have benefits; two IBLAZE survey respondents reported learning about IBLAZE from an old episode that had not been replaced by a more recent episode.

2) Quality Control: Quality issues prevent software development tools from being discovered by preventing them from appearing in a episode in the first place. While authors may be primarily motivated to write an episode to boost the user base of their tool, editors are motivated to publish episodes that increase the productivity of Google developers. When authors are not able to convince editors of their tool's merit, editors can reject the episode. Editors reported rejecting proposed episodes because the tools described in them had too few users and thus had not established the tool as best practice. They also reported that rejected authors sometimes complained, asking how could they establish the tools as a best practice without having a wider set of users. Although editors regarded themselves as guardians of quality and protectors of developers' best interests, they also noted that over the history of TotT, the supply of proposals for episodes waxed and waned, enabling them to be more selective at times and forcing them to accept some lower quality episodes at others.

Interviewees noted that tools are scrutinized before their episodes are published. For example, GENERATEDOC's authors noted that the editors complained that the tool was not yet good enough because the tool should not be a command-line tool run by individual developers, but instead should be available as a service that displays documentation on demand. The authors agreed that a service was likely the optimal architecture for this tool. However, to convince the editors that an episode about GENERATEDOC was worthwhile, the authors solicited and received testimonials from influential developers managing Google-wide C++ and Java ecosystems.

3) Other Discovery Challenges: The jokes shed light on several viewpoints on TotT not expressed by interviewees. One joke noted that episodes are overly idealistic about software engineering in practice; as a consequence, advice given in an episode may not be as as straightforward to apply as advertised. Another joke noted that the bathroom is a socially awkward place to spend any significant amount of time reading. Another complained of episodes being posted too far from the toilet to be read.

Readers who used IBLAZE noted three other challenges that prevented them from discovering IBLAZE from TotT:

- *Team Incompatibility.* Several respondents said IBLAZE was not useful to the team they were on.
- *Technology Incompatibility.* Some respondents used an old technology that did not work with IBLAZE
- *Inability to Appreciate Usefulness.* Some respondents noted that they didn't find IBLAZE useful at the time, because they were new employees, didn't find the tool provided a big productivity gain, didn't find the episode's presentation compelling, or didn't fit with their development style.

C. Other Influences on Tool Discovery

We also found that the usage or lack of usage of software development tools after TotT is partly a consequence of the design of the tool itself, especially in terms memorability, trialability, breadth of applicability, and usability.

1) Memorability: While TotT has an advantage over social media in that episodes are read in a fairly information-sparse environment, its episodes exist across the gulf of physical and mental space that separates developers' work environments from their private ones. The jokes commented on this tension, exultating the victory of retaining a URL on the journey back to one's workstation and lamenting over the difficulty in doing so.

Three tool authors believed the memorability of their tool name influenced usage. IBLAZE's author attributed the success of promoting his tool in TotT to the tool's high memorability, because the name 'IBLAZE' is a one-letter addition to the standard build tool, BLAZE. On the other hand, CHANGETIMEZONE's author attributed the tool's lack of wide usage to readers forgetting about it immediately after they returned to their desks from reading about the tool in the restroom. SANDBOXMANAGER's author stressed memorability further, arguing that it was critical for some kinds of tools such as SANDBOXMANAGER, which may not be used the same day that it's read about in the restroom. Instead, he argues, developers read about the tool, then must recall it later when they have a problem that the tool solves.

2) Trialability: How easy it was to try a tool also may have influenced usage, a property called *trialability* in diffusion of innovations theory [37]. Based on the number of questions that developers asked about UIDIFF on mailing lists, the author initially felt that the episode had a lot of impact, but was surprised by how little impact the episode had on users in retrospect. The author attributes this to difficulty in trying the tool, which required the user to collect a substantial amount of up-front information to determine if the tool would be useful in their context and to configure it fairly extensively. Likewise, authors of ESTIMATERESOURCES attributed a lack of wide usage to low trialability because the tool takes a while to set up.

3) Other Properties: Participants attributed several other properties to their tools' usage rates. For CHANGETIMEZONE, although Google has dozens of offices globally, many developers work at the headquarters or in its time zone. So, many developers who would see CHANGETIMEZONE's episode would not find it relevant. VERIFYDETERMBUILD's author acknowledged that usability of his tool was a challenge; the tool's output is verbose and does not present a solution to the problem. PYTHONFORMATTER's author believed that user expectations influenced usage rates. While the number of users is sustained beyond the episode, it does not grow, in contrast to CLANG-FORMAT. The author attributes this to users expecting that PYTHONFORMATTER worked like CLANG-FORMAT: at the time of the episode, it was substantially more limited and did not reformat Python files but instead only fixed a small set of lint errors. Finally, the author of COVERAGE noted that his advertisement was successful because it was conductive to the TotT format, which forces authors to concisely state why their tool is useful from a non-toolsmith's perspective. He noted that COVERAGE is simple enough that it could easily be explained quickly, so the episode format was ideal. Editors agreed that brevity and motivation are critical factors in the success of TotT, and that working with authors to draft and refine their episodes is a time-consuming process, often because developers' initial drafts lack these properties.

D. TotT Versus Other Techniques

1) Social Media: Some authors used internal social media as well as Testing on the Toilet to promote their software development tools. However, TotT consistently resulted in a larger increase in the number of developers using the tools. Authors mentioned advertising three tools via social media:

- Visible at the left of Figure 2a, the episode author promoted CLANG-FORMAT via a social networking site and a mailing list. While the raw number of users gained from TotT was higher, the social media campaign yielded a higher proportion of new users than TotT. Moreover, social media showed 10% week-over-week growth (about 4 new users per week) in the weeks following the social media posts, whereas the episode was followed by a 6% week-over-week growth (about 23 new users per week). CausalImpact confirms the social media campaign yielded an increase in tool usage ($p = .002$) with a higher relative impact (529%) than the episode (264%). While social media caused what appears to be substantial initial try-outs of the tool followed by a substantial drop in the user base, the episode appears to show sustained growth even after the episode was superseded. One reason may be that the version promoted by the social media release was likely buggier than the version promoted by TotT.
- IBLAZE's creator promoted his tool through internal social media, which was shared by one well-known developer within Google about four months before the episode. That post helped the tool grow from a peak of 20 daily users to a peak of 36, growth that CausalImpact reports as non-significant ($p = .119$).
- For VERIFYDETERMBUILD in Figure 2f, the fourth peak near the end of the publication period (that is, near the dotted vertical line) may be due to a social media post from a developer with about 3300 followers today. The post mentioned VERIFYDETERMBUILD as a means to complain about a compiler that always produces different binaries. Another social media post about VERIFYDETERMBUILD, 9 months before the episode, yielded a small but statistically significant boost (0.15 users per day, $p = .02$), though the poster in this case has around 350 followers today.

These results suggest that social media posts can increase the number of developers using tools, but the magnitude of the increase depends on several factors, including how many followers the developer has. In contrast, because each episode is distributed by the same group of volunteers, TotT's influence is not limited by the individual developer's personal network. But as we discuss in Section V-B2, maintaining the influence of TotT is challenging.

2) Other Advertising Mechanisms: Developers used other ways of advertising their tools. In the case of EmulatorSettings, we notice the three peaks several months before the episode in (Figure 2g). As the author suggested and follow-up investigation of emails confirmed, the tool creators sent out announcement emails to mailing lists during the week the first two peaks appeared (though we have no explanation for the peak the week after). These announcement emails were sent to several mobile development and testing mailing lists. These emails caused growth from about 8 daily users to about 30 daily users, peak-to-peak, a significant increase ($p = .03$). A major difference between the TotT and mailing list promotions is that the mailing lists were more targeted at people who were likely to use the tool (that is, mobile developers), compared to the much more widely spread episode, which includes many developers who do not develop on Android. Interestingly, after the mailing list promotion, daily users drop to about pre-intervention levels. In comparison, after the episode, daily usage remained high. One possible explanation is that emails are more ephemeral than episodes; as emails are forgotten, they are deleted or buried under more emails, but when episodes are not updated, they can linger in bathrooms for indefinite periods, educating developers who pass through.

The creators of EstimateResources unsuccessfully advertised their tool through a poster a few months before the episode. Specifically, they promoted the tool at an annual project fair at Google's largest European software development office. The one-day tool usage around this date shows no appreciable impact from the poster, confirmed by CausalImpact ($p = .27$). The authors attributed this lack of success to the fact that their poster had poor placement and that the product fair's audience was much broader than the tool's potential audience.

When we asked developers who didn't learn about iblaze from the episode where they did learn the tool from, they mentioned several other advertising mechanisms through which they discovered iblaze. Many respondents reported discovering iblaze from internal Google tutorials, which focused on other technologies, but which made iblaze an integral part of the activities. Three respondents learned about iblaze from the iblaze badge, a small hyperlinked graphic displayed on iblaze-using developers' profile pages in the employee directory. While the effectiveness of these interventions relative to TotT is unclear, these results suggest they can be nonetheless effective to some extent.

3) Peer Learning: Developers, including several respondents to our survey about iblaze, reported learning about tools from peer developers. Learning tools from other developers was also mentioned by the author of the PatchSearch episode, where usage of this tool increased gradually in the years following the episode. In contrast, the EmulatorSettings tool has seen relatively flat usage over the past 1.5 years. Murphy-Hill and colleagues provide a plausible explanation [35]: EmulatorSettings is not easily visible to others because it runs entirely on one developer's workstation, whereas PatchSearch is commonly integrated into scripts checked into version control, which are then visible to the entire company.

Even if TotT does not itself raise awareness of a tool, an episode can nonetheless be supportive of other forms of tool discovery, such as through peer learning or social media, but being a self-contained module of information that developers can refer back to later. In the case of peer learning, recommending developers can refer the learner to an archived episode, and that episode can serve as a self-contained witness to the tool's usefulness.

E. Other TotT Motivations

In beginning this study, we assumed that authors' motivation for publishing their tool in Testing on the Toilet was to increase usage. However, we found three other factors that motivated authors to share tools.

1) Preventing Reinvention by Establishing Usefulness: An author noted that his desire to publish an episode about SandboxManager was not so much about getting more people to use SandboxManager, but instead to get other teams in Google to stop building tools that duplicate SandboxManager. He had personal experience with this; a team that he had just moved to had implemented their own version of SandboxManager. The team was resistant to using SandboxManager, arguing that it was specialized for some other product within Google, a variant of the Not Invented Here syndrome [25]. To evaluate how widely useful SandboxManager was beyond the product it was built for, the author wrote an episode. The editors of TotT acted as an informal committee to weigh in on whether SandboxManager was widely useful.

2) Justifying Increased Investment: The PythonFormatter episode provided a critical mass of users that justified increased investment in the tool's development. The episode author was able to spend development effort replacing the core of the tool, increasing its usefulness.

3) Inspiring Future Tools: Interviewees reported that a major goal of releasing an episode about GenerateDoc was to demonstrate the benefits of automatic documentation to the documentation infrastructure team. In fact, the team took notice of GenerateDoc and subsequently began work on their own approach.

VI. Discussion and Future Work

Overall, because 11 of 12 tools showed statistically significant increases in the number of daily unique users after each Testing on the Toilet was published, we conclude that TotT was effective for encouraging developers to try new tools at Google. Compared to software engineering recommender systems published in the research literature (e.g. [33], [40]), TotT has several advantages: it does not require substantial infrastructure or technical implementation, it does not require user targeting and customization, and it can be used to showcase a variety of tools. However, it has some disadvantages: it requires an organization-wide network of people to distribute episodes, it requires the writing of concise and convincing prose on a per-tool basis, and it requires developers to be willing to have a private space (restrooms) used for purposes other than their core function.

In Section V-D, we saw that TotT increased the raw usage of tools more substantially than social media posts. While the past 15 years in consumer advertising has shifted towards being digital, personalized, and interactive [29], our results are a reminder that physical interventions can still play an important role in shaping developer behavior. While social media can reach a wide variety of users, the physical form and context of TotT has advantages as well. First, apart from one's primary business is a restroom, the generally low information density in a restroom means developers' attentional demands are relatively low. Second, unlike in a digital setting, a developer's attention is sustained for a fixed period because the biological processes that developers engage in while reading TotT cannot be substantially sped up. Third, the regularity of biological needs ensures repeated exposure to episodes. Fourth, the one-episode-per-week pacing ensures that developers aren't deluged by many tools in a short amount of time. Finally, because TotT is written by developers, a medium that might otherwise be an invasion of personal space is instead a quirky—but, as we have shown, effective—medium to spread knowledge about tools.

Would TotT be equally effective if posted in other places? Lactation rooms are similar to restrooms in that they both are used for biological needs, but lactation rooms serve a narrower user base. Nap rooms can potentially serve a wide audience, but are typically too dark for reading in our experience. Cafeterias have similarities as well, but at least at Google, time spent in cafeterias is culturally considered time to socialize with coworkers. And while Google employees have experimented with other spaces for posting fliers [6], restrooms have remained the space of choice for TotT.

Participants complained of irregular coverage of episodes in Section V-B1. While theoretically bathroom-based fliers provide an equal opportunity to access TotT information because "everybody poops" [18], the irregular coverage across buildings practically means that not every developer has an equal likelihood of encountering episodes. Moreover, our experience as developers is that women are less likely to encounter episodes than men. This seems statistically likely; since TotT was distributed by volunteer developers and more developers are men, the likelihood of a woman volunteering to post episodes in women's restrooms in a given building is lower than that of a man, all else being equal. Moreover, encouraging more women to volunteer in an effort to remedy the problem imposes a *minority tax*, "the burden of extra responsibilities placed on [minorities] in the name of diversity" [36]. We believe that we have resolved this recently by making flier distribution a regular job responsibility of the facilities staff. The minority tax problem could also be alleviated by replacing gendered restrooms with gender-neutral restrooms.

While we have demonstrated that TotT generally increased tool use at Google, we readily admit that we have not shown that TotT consistently drives long term tool adoption or that adopting the tools improves software quality or developer productivity. To understand whether TotT (or any other intervention, for that matter) drives long-term adoption, other measures of adoption are needed. Future researchers should consider flexible metrics for software tool adoption; doing so may help us close the loop and better understand the diffusion of innovations for software engineering tools. Such metrics would ideally be validated for construct validity and used in triangulation with one another. Likewise, even if we were to show that developers adopt tools over the long term, it remains an open problem how one would quantify whether developers' productivity or code is improved as a consequence of tool use.

VII. Conclusion

As the number and breadth of available tools grow, software developers need effective techniques to keep abreast of useful tools. Over the last decade, Google has been using a paper-based technique, Testing on the Toilet. Analyzing a large data set with six years of tool logs using causal inference, then contextualizing that usage with perspectives from authors, editors, and readers, we demonstrated TotT's ability to help developers to discover new tools.

VIII. Acknowledgments

Thanks to anonymous interviewees and reviewers, Thomas Adamcik, Sameer Ajmani, Alexander Buccino, Chris Conway, Jeff Cox, Adriano Cunha, Damien Desfontaines, Sanjeev Dhanda, Lorenzo Dini, Marko Ivankovic, Ivan Janicijevic, Daniel Jasper, Max Kanat-Alexander, Thomas Knych, Raphael Menderico, Eric Nickell, Maria-Hendrike Peetz, Viola Petra, Antoine Picard, Rob Pike, Rachel Potvin, Martin Probst, Christoph Raupach, Stefan Sauer, Rob Siemborski, Justin Smith, Bill Wendling, JC van Winkel, and the CausalImpact team.

References

[1] https://figshare.com/s/4b9e5bd8a6ab17a9f040.

[2] Lavnotes. Published by Sheridan Libraries at Johns Hopkins University, 2012. http://old.library.jhu.edu/about/news/lavnotes/index.html.

[3] Bathroom reader. Published by Harvard Law School Library, 2017. https://www.scribd.com/collections/2835528/Bathroom-Reader.

[4] Clangformat. LLVM Documentation, 2017. https://clang.llvm.org/docs/ClangFormat.html.

[5] Schibsted testing on the toilet. Published on Twitter, 2017. https://twitter.com/Schibsted_Eng/status/862239498100518912.

[6] Mike Bland. Testing on the toilet, 2011. https://mike-bland.com/2011/10/25/testing-on-the-toilet.html.

[7] Julie Bort. How Google convinced programmers to do something called 'testing on the toilet', 2014. http://www.businessinsider.com/googles-famous-testing-on-the-toilet-2014-12.

[8] Kay H Brodersen, Fabian Gallusser, Jim Koehler, Nicolas Remy, and Steven L Scott. Inferring causal impact using Bayesian structural time-series models. *The Annals of Applied Statistics*, 9(1):247–274, 2015.

[9] Dustin Campbell and Mark Miller. Designing refactoring tools for developers. In *Proceedings of the 2nd Workshop on Refactoring Tools*, pages 1–2, 2008.

[10] Tse-Hsun Chen, Weiyi Shang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. CacheOptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 666–677. ACM, 2016.

[11] Kurt Christensen. Unit testing tips from Google, 2007. https://www.infoq.com/news/2007/04/google-testing-tips.

[12] Alistair Cockburn and Laurie Williams. The costs and benefits of pair programming. In *Proceedings of the Conference on Extreme Programming*, pages 223–247, 2000.

[13] John W Creswell and Dana L Miller. Determining validity in qualitative inquiry. *Theory into practice*, 39(3):124–130, 2000.

[14] Andrea Di Sorbo, Sebastiano Panichella, Carol V Alexandru, Junji Shimagaki, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the Symposium on Foundations of Software Engineering*, pages 499–510. ACM, 2016.

[15] Robert G. Fichman and Chris F. Kemerer. The illusory diffusion of innovation: An examination of assimilation gaps. *Information Systems Research*, 10(3):255–275, 1999.

[16] Stephen R. Foster, William G. Griswold, and Sorin Lerner. WitchDoctor: IDE support for real-time auto-completion of refactorings. In *Proceedings of the International Conference on Software Engineering*, pages 222–232, 2012.

[17] Xi Ge, Quinton L DuBose, and Emerson Murphy-Hill. Reconciling manual and automatic refactoring. In *Proceedings of the International Conference on Software Engineering*, pages 211–221, 2012.

[18] Tarō Gomi. *Everyone poops.* Kane/Miller Book Pubs., 2001.

[19] Sara Kehaulani Goo. Building a 'Googley' workforce, 2006. http://www.washingtonpost.com/wp-dyn/content/article/2006/10/20/AR2006102001461.html.

[20] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 649–658, 2009.

[21] Karsten C Hofmann. *Advertising in restrooms.* PhD thesis, Portland State University, 1988.

[22] Juhani Iivari. Why are CASE tools not used? *Communications of the ACM*, 39(10):94–103, 1996.

[24] Lance P Kaltenbaugh, Janel C Molnar, Wesley N Bonadio, and Brittany L Dorsey. A study on restroom advertising and its effect on awareness of campus recreation programs. *Recreational Sports Journal*, 35(1):3–11, 2011.

[23] Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K Smith, Collin Winter, and Emerson Murphy-Hill. Advantages and disadvantages of a monolithic repository: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 225–234. ACM, 2018.

[25] Ralph Katz and Thomas J Allen. Investigating the not invented here (NIH) syndrome: A look at the performance, tenure, and communication patterns of 50 R& D project groups. *R&D Management*, 12(1):7–20, 1982.

[26] Dominik Lehmann and Donald J Shemwell. A field test of the effectiveness of different print layouts: A mixed model field experiment in alternative advertising. *Journal of Promotion Management*, 17(1):61–75, 2011.

[27] Frank Linton, Deborah Joy, Hans-Peter Schaefer, and Andrew Charron. OWL: A recommender system for organization-wide learning. *Educational Technology & Society*, 3(1):62–76, 2000.

[28] Yepang Liu, Chang Xu, Shing-Chi Cheung, and Valerio Terragni. Understanding and detecting wake lock misuses for android applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 396–409. ACM, 2016.

[29] Matthew Lombard and Jennifer Snyder-Duch. Digital advertising in a new age. *Digital Advertising: Theory and Research*, page 169, 2017.

[30] Michael Mackert, Ming-Ching Liang, and Sara Champlin. "think the sink:" preliminary evaluation of a handwashing promotion campaign. *American Journal of Infection Control*, 41(3):275–277, 2013.

[31] Carlos Maltzahn. Community help: discovering tools and locating experts in a dynamic environment. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 260–261, 1995.

[32] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. CommunityCommands: command recommendations for software applications. In *Proceedings of the Symposium on User Interface Software and Technology.*, pages 193–202, 2009.

[33] Emerson Murphy-Hill, Rahul Jiresal, and Gail C. Murphy. Improving software developers' fluency by recommending development environment commands. In *Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 42:1–42:11, 2012.

[34] Emerson Murphy-Hill and Gail C. Murphy. Peer interaction effectively, yet infrequently, enables programmers to discover new tools. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pages 405–414, 2011.

[35] Emerson Murphy-Hill, Gail C Murphy, and Joanna McGrenere. How do users discover new tools in software development and beyond? *Computer Supported Cooperative Work*, 24(5):389–422, 2015.

[36] José E Rodríguez, Kendall M Campbell, and Linda H Pololi. Addressing disparities in academic medicine: what of the minority tax? *BMC Medical Education*, 15(1):6, 2015.

[37] Everett M. Rogers. *Diffusion of Innovations.* Free Press, 5th edition, 2003.

[38] Leif Singer. *Improving the Adoption of Software Engineering Practices Through Persuasive Interventions.* PhD thesis, Gottfried Wilhelm Leibniz Universitat Hannover, 2013.

[39] Edward K Smith, Christian Bird, and Thomas Zimmermann. Build it yourself!: Homegrown tools in a large software company. In *Proceedings of the 37th International Conference on Software Engineering*, pages 369–379. IEEE Press, 2015.

[40] Petcharat Viriyakattiyaporn and Gail C. Murphy. Improving program navigation with an active help system. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*, pages 27–41, 2010.

[41] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. Quantifying developers' adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 260–271. ACM, 2015.

[42] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. Social influences on secure development tool adoption: Why security tools spread. In *Proceedings of Computer Supported Cooperative Work and Social Computing*, pages 1095–1106, 2014.

[43] Robert K Yin. *Case Study Research.* Sage, 1994.