Proceedings of the 54th Hawaii International Conference on System Sciences | 2021

A Behavioral Approach to Understanding the Git Experience

Genevieve Milliken
New York University
genevieve.milliken@nyu.edu

Sarah Nguyen
University of Washington
snguye@uw.edu

Vicky Steeves
New York University
vicky.steeves@nyu.edu

Abstract

The Investigating and Archiving the Scholarly Git Experience (IASGE) project is multi-track study focused on understanding the uses of Git by students, faculty, and staff working in academic research institutions as well as the ways source code repositories and their associated contextual ephemera can be better preserved. This research, in turn, has implications regarding how to support Git in the scholarly process, how version control systems contribute to reproducibility, and how Library and Information Science (LIS) professionals can support Git through instruction and sustainability efforts. In this paper, we focus on a subset of our larger project and take a deep look at what code hosting platforms offer researchers in terms of productivity and collaboration. For this portion, a survey, focus groups, and user experience interviews were conducted to gain an understanding of how and why scholarly researchers use Version Control Systems (VCS) as well as some of the pain points in learning and using VCS for daily work.

1. Introduction

An important part of the scholarly record is unstable. While there are many initiatives to incentivize, support, and publish research outputs such as data and electronic notebooks/field notes, these are noticeably lacking for code and software. This is compounded by the fact preservation and access workflows are less systematized for code and software curation than other types of research outputs (e.g. A/V media, manuscripts). Such a lack is particularly problematic because a multitude of disciplines, from the sciences to the humanities, write or use code in their academic research. Further, many researchers use Version Control Systems (VCS) such as Git as well as hosting platforms like GitHub to collaborate, version, and publish code openly.

Many of the most popular Git Hosting Platforms (GHP), however, do not have a long-term preservation plan and make no guarantees about the long-term availability of work on their platforms. This is particularly problematic because the most commonly used GHPs are commercial companies and are not in the business of preservation. Further, these companies are not immune to closures and such cessations have, in fact, already occurred and include Google Code in 2016 and Gitorious in 2014 as well as many other open source code forges [1]. In 2020, BitBucket also suspended support for Mercurial repositories, further forcing developers to "go mainstream" and use Git and GitHub in their workflow [2].

In addition to the dependency and precarity of these platforms, how and why scholarly researchers use Git and GHP during their research process remains opaque and understudied. These platforms allow for sharing, collaboration, and even scholarly interactions such as peer review and annotations of code, though it comes with steep technical and social learning curves. A contributing factor to this may be that these platforms were created for software development and not academia, making the learning, use, and support of Git idiosyncratic at the project, department, and institutional levels.

To date, there has been no large-scale or in-depth study of how researchers use these tools. To fill this gap, the Investigating and Archiving the Scholarly Git Experience (IASGE) project seeks to understand scholars' engagement with Git as a VCS and GHP as a mechanism for scholarly communication and dissemination. Through a behavioral studies approach, this paper takes a deep look at what code hosting platforms offer to researchers in terms of productivity and collaboration and uses a survey, focus groups, and user experience interviews to gain an understanding of how and why scholarly researchers use VCS as well as some the pain points involved in learning and using VCS. Our main findings are that a wide variety of researchers, from the humanities to sciences, use Git, but often lack a mental model for understanding it. This situation results in a poor understanding of the tool and suggests that support for learning and adopting Git, as well as training in best practices, is needed.

2. Background

There is a need in software development to track changes over time, and this role is fulfilled by VCS. Also known as revision control systems or software management systems, VCS allows users to track changes, compare different versions, and merge changes to a codebase collaboratively over time. Version control is done on a repository, which is a directory of source code files with an underlying data structure that contains the metadata for all tracked files, with a historical record of all files. Depending on whether the VCS in use is centralized (e.g. Subversion, CVS) or distributed (e.g. Git, Mercurial), the repository may contain varying amounts of information. Distributed VCS became more popular because each mirror of the repository (e.g. on each developer's computer) is considered "equal," and cannot be automatically overwritten by another, allowing users more flexibility and the ability to work without fear of corrupting anyone else's progress. Git, the VCS explored in this paper, was created as a result of a professional conflict during development of the Linux kernel. Previously, the open source project had been using BitKeeper (a proprietary centralized VCS) for source code management [3], [4]. However, the copyright holder of BitKeeper rescinded access to the software, believing that some were trying to reverse engineer BitKeeper's protocols, and in response, Git was created.

Since then, Git has emerged as the most popular VCS [5], due in large part to its scalability and support of non-linear code development as well as a 'toolkit' design that facilitates large collaborative projects with many merges. Web-based platforms called "forges", now more broadly known as "source code hosting platforms," are important for understanding VCS. In Free and Open Source Software (FOSS), a "forge" is a web-based collaborative software development platform that includes not only support for repository management, but also services and integration such as mailing-lists, wikis, bug tracking, and code review. The first third-party hosting platform for code repositories was Helix TeamHub in 1995, which worked for Perforce, an early VCS. More platforms have grown to include what the authors will refer to as "Git Hosting Platforms" (GHP) that include elements of forges, but forefront or exclusively support for Git (e.g. GitHub). Just as not all squares are rectangles, not all forges are GHP, but all GHP are forges.

While VCS have been used in software development since 1975, it has recently become popular in academic institutions. This project is interested in how Git is used in academia and how GHPs are used as platforms for scholarly engagement. The population we are interested in studying are all involved with a range of scholarly endeavors (e.g., graduate studies, grant-funded labs, research in areas of expertise, etc.) and include students, faculty, instructors, researchers, and staff who develop and maintain software and code.

A key component of this study is to put version-controlled research software into a wider context and highlight its place as a scholarly research artifact alongside other non-manuscript outputs such as lab notebooks, datasets, executable papers, data visualizations, and digital humanities projects. Recently, data has received the most attention of these computational outputs due to the mandate that federally-funded research be made freely available [6] and the application of the FAIR principles for research code [7]. Calls for data sharing [8], professional organizations such as the Data Curation Network [9], and professional roles such as Data Librarians [10] have also helped to further foreground the importance of data. Data repositories and generalist repositories have also facilitated the preservation and access to datasets and have made them more accessible to the public. Unlike data, however, research software has received far less attention and has fewer institutionalized roles dedicated to its maintenance and curation of these complex digital objects. Software is particularly difficult to reproduce and preserve due to multiple files/directories, use of packages and libraries, hardware, and operating systems dependencies, as well as its required documentation and metadata for others to use.

Our interest, however, is not solely with the source code but also the contextual information around it. GHP are also quite complex and come equipped with a set of ancillary tools within each repository that help developers and academics alike organize, communicate within, and disseminate their projects. These include features such as wikis, issues (which often act as discussion threads), and pull requests. We will refer to these tools and the scholarly interactions that take place within them as "scholarly ephemera." These contextual materials are evidence for understanding the history of a repository, how one repository might relate to another, how members of each repository branch out and form networks, and how this information can be used to track derivatives of current work. Currently, scholarly ephemera is not being archived with the same breadth as source code.

In summation, our work described below is centered on the following premise: that scholarship is being produced on GHPs and GHPs are geared towards software development, rather than academia. This results in a steep learning curve, a weak academic support system, and a lack of essential scholarly features.

To explore this, our primary research question for this paper is: when learning and adopting Git and GHP in a scholarly context, what are the barriers to entry, what are the benefits of acquiring the skills needed to use them, and what support systems exist (or should exist) to accommodate this effort?

3. Methodologies

Our approach to studying the experiences of researchers using Git and GHP is drawn from the social sciences. The sections below detail the methodologies for the active components to our project, which include a series of focus groups, a broad survey, and interviews. We employed a multi-method study in order to understand scholars' use of Git and GHP; starting with a series of focus groups (producing qualitative data), followed by a broad survey (producing quantitative data), and finishing with a set of in-depth, user-experience interviews (also producing qualitative data). Using three different research instruments help us to triangulate our data in order to verify findings by incorporating three different viewpoints [11], and "for cross-checking, or for ferreting out varying perspectives on complex issues and events" [12]. All materials underlying this study are available in the Supplementary Files section.

The focus groups consisted of synchronous in-person discussions aimed at better understanding "minimal users." This type of user can be defined as scholars who are aware of Git, or who have taken a workshop or course to learn the basics of Git and GHP, but have not incorporated it into their research or scholarly process. These focus groups included three group sessions with a total of 12 participants (e.g. students, faculty, researchers, etc.) who have previously taken "Introduction to Git and GitHub" classes. Recognizing minimal users as a key persona among the scholarly Git experience population is significant in understanding scholars' behavior with Git and GHP. As noted by Glassey, educators are often surprised by students' overwhelming enthusiasm for adopting VCS, but also "discovered that they lacked understanding about the system or having confidence in their ability to use it effectively beyond the course" [13]. The IASGE project is interested in the gap between encountering and adopting Git and the focus groups are a means to understand it more fully.

The focus group analysis employed a grounded theory approach, with inductive and discourse analysis. This gave an understanding of participants' social and cognitive psychology, their memory and attitudes related to learning VCS, and their experiences with Git and GHP [14]. In order to create and refine our qualitative codes, transcripts were uploaded to Taguette, a free and open source qualitative analysis tool, for individual "closed" coding [15]. Then, the transcripts were switched and re-coded, between two of the authors, to pinpoint social interactions that may have been overlooked during the first round of coding [16]. Taguette was used for collaborative review to consolidate findings and themes. Attributes and terms from the qualitative coding can be found in the codebook within the Supplementary Files. Themes from tagging, thematic coding, and reviewing shed light on the usability of Git and GHP and their implications for reproducibility of research code and the survivability of its scholarly ephemera.

Information from the focus group discussions also guided the topics included in the subsequent broad survey. This quantitative phase of the behavioral study tested the generalizability of our findings by surveying self-selecting scholars and researchers, ranging from minimal to advanced Git users. We had 371 participants that met our criteria for inclusion in analysis, which is that they are currently working in academia and writing code and using version control for that code. The survey includes 54 questions that were divided into eight sections, including branching logic based on answers given. These sections helped to identify some basic demographics, the level at which they use Git in their scholarly workflow, and gathered information about participants' previous experiences in learning and/or teaching Git. The survey was hosted and delivered through an institutional license for the survey web application Qualtrics. The survey was designed to appeal to a broad audience and was distributed through listserv, forum announcements, social media, and to individuals involved with open science, open source software, and digital preservation communities. The survey data illustrates the wide spectrum of user proficiency levels across learning, adopting, teaching, and actively using Git throughout their scholarly workflow[1]. In general, the survey's variables were either nominal or ordinal since they may or may not have the ability to be ordered or ranked, nor hold measurable distances from each other [17]. The majority of the data analysis included frequency distributions and cross-tabulation [18].

To recruit for the project's in-depth interviews, the survey concluded with a question asking participants if they would be willing to participate in a scenario and task-based interview at a later date. 110 agreed, nearly 90 consented (most of whom rated their skills with Git as intermediate or above), and the authors ultimately conducted 41 one-hour interviews over the course of

1 The focus groups, survey, and interviews received IRB approval, #IRB-FY2019-3399.

June 2020. These participatory user sessions centered around asking participants to perform a series of tasks such as cloning a repository, making a pull request, and resolving a merge conflict. Due to COVID-19 restrictions, qualitative analysis for the interviews has recently started in earnest. Although preliminary, we will select and include salient, recurring quotes/themes from an initial review of the interview transcripts to triangulate emerging patterns. We will attribute quotations, from both the focus groups and the interviews, to individuals using aliases and the relevant transcript file name (e.g. Shelly Deepak, 20200623_transcript_001).

4. Findings

In assessing the preliminary results from the behavioral study of IASGE, we have identified key areas relevant to understanding the "scholarly Git experience." Our findings indicate that studying Git and GHP from the perspective of usability allows greater insight into the experiences of learners at both the student and professional levels. Usability studies provide insights to the pain points, learning curves, "friendliness", and effectiveness of Git for individuals [19]. For the purpose of this paper, usability is viewed as a determining factor for the adoption and use of Git and GHP.

4.1. Understanding Users

The studies of both minimal users and advanced users of Git and GHP have shown that there is a range of adoption rates for VCS in both classroom and professional settings. There are many limitations to learning Git, and it can be overwhelming when trying to also learn object-oriented programming and syntax in tandem with VCS. As Jennifer Bryan noted, "Git was built neither for [classroom or scholarly] usages [...] nor for broad usability", and so our usability research is concerned with how scholars adopt and adapt Git and GHP through their daily workflow, including successes and failures encountered in the research process [20].

The second section of the project's survey was "Learning Git" (Questions 14:22) and asked participants to share when, why, and how they first learned Git (e.g. who taught them, how difficult it was to learn, what are some re-learning strategies, and favorite resources). Unsurprisingly, 42.6% of the survey respondents noted that their primary reason for learning and adopting Git was a need to version their software. This was followed by 34% of the respondents who adopted Git because collaborators were already using it for a project. Needing to use Git for a course, to upgrade from one VCS to another, keeping up with changing technology standards, work requirements, reproducibility, and backup were the remaining reasons ranked from most common to least common. Seen through these categories, the incentives to adopt Git were less affected by personal motivation (e.g. backups) and more due to external circumstances such as a need to work on a collaborative project with others or as required in a course.

These are notable findings in relation to the literature on VCS used in the classroom. For instance, Bacharakas reports that 70% of university students who wanted to contribute to open scholarship hosted on GHP were not able to because they did not know where to start [21]. It was common for graduate students from the focus group sessions to express how tutorials and workshops made it clear that Git would be beneficial but they still came out unsure on how to integrate it into their workflows (20200313_transcript_001, 20200410_transcript_002). Isomöttönen and Cochez found students in a project-based course encountered several common difficulties related to using Git in a collaborative environment [22]. For instance, students often only use the basic Git commands (e.g. git commit -a), avoid experimenting with advanced commands, reclone repositories when conflicts occur, confuse Git and bash commands, and often lack a proper mental model for Git. One contributing factor to this may be similarities of commands and the order in which they need to be executed. In several focus groups and interviews, participants mentioned that they found Git's terminology confusing (20200410_transcript_002, 20200710_transcript_003), leading to uncertainty around syntax and structure that beginners may find idiosyncratic.

4.2. Lacking a Mental Model

Many of the experiences expressed in Isomöttönen and Cochez were not just evident in our focus groups, but also in the in-depth interviews with more experienced Git users [22]. For instance, a merge conflict that the participant needed to resolve was intentionally built into a series of scenario and task-based exercises. While many participants resolved this through Git commands and/or through extensions to IDEs (e.g. Visual Studio Code and Atom had GUI buttons to keep your vs. their changes), a subset deleted and re-cloned the repository. For instance, a postdoctoral student fell into the habit of deleting and recloning the test repository to avoid the merge conflicts altogether (Anna Jakobson, 2020-07-01_00031). This behavior was echoed by another interviewee, who admitted that, even after three and a half years of experience with Git, they still "ha[ve] to go back […] to burn down a repo, delete it, and restart it" (Tanel Paasuke, 2020-06-22_00001). The same interviewee also noted that they felt they lacked a "robust mental model for how this thing is actually working [which] prevents me from figuring [Git] out on my own." They also stated that "Git tends to leave me questioning my understanding of things" (Tanel Paasuke, 2020-06-22_00001). Not all participants struggled equally with this, however. For example, a bioinformatics doctoral candidate, described a mental model for Git as an "idea of your copy [...] and someone else copying [... both which] negotiate and create something [different] from those two separate things" (Leili Mark, 2020-07-03_00037).

Conceptual or syntax-based confusion are not the only types of complications that are encountered. Complications also occur when scholars confuse Git with GHP. For instance, when asked "What are the top three things that I need to know to use this platform with you?" (Focus Group Question 7). A number of interviewees quickly responded with "commit, push, pull", giving Git-specific commands rather than speaking about collaborative features on the web-based GHP. In this case, basic Git commands were expressed as priority rather than the built-in collaborative features on GHP such as continuous integration or discussions on issue boards. Another example of this is when a minimal user from a focus group session admitted that "the biggest barrier for [them] is to understand the whole procedure of using GitHub", particularly the difference between pull and push (Peyman Meskini, 20200410_transcript_002).This further demonstrates confusion between specific commands and the platform itself. Similar to the mental model, the distinction between Git and platforms, like GitHub or GitLab, are often unclear.

A correct understanding of Git often requires users to mentally picture what a particular command, such as pushing, pulling, branching, and merging, are doing to a repository and what, if any, is their effect on the structure of the repository. Minimal users from the focus group sessions agreed that it is often difficult to mentally visualize or conceptualize these aspects of a Git repository, and how branching works. An engineering master's student, for instance, noted the lack of clarity when "learning [from] the web, internet, [or] on my own [...] looking at those descriptions, [it was] hard to understand what they are talking about. I have to match the description to the picture of the workflow to make me understand more, more like straightforward for me" (Leili Mark, 20200313_FocusGroup01). Only recently have we seen a greater emphasis on providing students with better mental models for learning Git, such as a visualizing Git tool[2] and "Git For Ages 4 And Up" [23], in which tinker toys are used to visually and physically demonstrate Git branching and merging, as well as other learning and visualization tools[3].

In addition to a lack of a mental model, focus group participants and interviewees discussed confusion when learning Git alongside other computational tools. In reality, Git and GHP are often learned alongside computational languages like Python, R, and MATLAB. Since Git has a steep learning curve, it is common for students and professionals alike to become overwhelmed when learning other syntaxes in tandem. As one anthropology postdoctoral researcher noted, students have to be extra cautious when "figuring out how to prioritize [...] do I try to learn Python? Or [...] do I try to [...] enhance my R skills, or my MATLAB skills, or do I prioritize GitHub?" (Rahele Deljou, 20200313_FocusGroup_001). It is clear that lack of a mental model and learning many tools simultaneously creates a steep learning curve for any Git user. In this academic context, this hurdle is even steeper for humanities scholars who do not have prior programming foundations like those in the computer science and engineering disciplines. In both focus group and interview sessions, those who studied computer science or engineering prior to adopting Git, had exposure to the basics of the utility diff—initially created as a data comparison tool originally made for Unix, Plan 9, and Inferno operating systems [24]–[26]. This implies that the participant had experience with historical computing systems prior to using Git, giving insight on understanding how file versions are compared. Similarly, those who have previously learned and used other VCS during their undergraduate training, such as SVN, Mercurial, CVS, etc., had an advantage in understanding the infrastructure of versions and repositories that is primary to Git's mental model (2020-06-23_00003, 2020-06-26_00015, 2020-07-01_00033, 2020-06-29_00010, 2020-06-30_00024).

4.3. Learning Support

Self-directed learning was indicated as a method used by survey participants to learn or re-learn Git. Of those who answered the question, 51 out of 153 survey respondents claim to be self-taught versus through a formal classroom or workshop setting. 139 out of 371 of the total respondents attribute books and online

2 https://git-school.github.io/visualizing-git/

3 Examples include: https://github.com/jlord/git-it-electron#what-to-install, http://git-school.github.io/visualizing-git/, https://learngitbranching.js.org/

resources, commonly used for independent learning, as their primary source for instruction (Figure 1).

Many survey participants first started to use Git out of necessity (e.g. needed version control, collaborators were already using it, or required for a course. There is a need for more structured Git learning in order to help those that need version control and collaborative tools for active projects. Self-paced teaching paired with motivations and external pressures (e.g. using Git for collaborative projects), while also fostering a proper mental model, could be better managed by offering more avenues for iterative and project-based learning, which is gradually making its way into academic curricula. A computer science graduate student in a focus group, for instance, noted that they encountered Git in the classroom, but did not take the opportunity to comprehend Git at the time. However, they relearned Git when they had to do their graduation capstone project and needed to "provide [the code to] the professor" (Mahyar Shariati, 20200313_FocusGroup_001). Without realistic applications and project-based learning opportunities, students often do not or cannot prioritize learning Git until they have to. There is value in incorporating Git in and out of the classroom to ensure students remember to highlight the benefits of version control throughout their career [27].

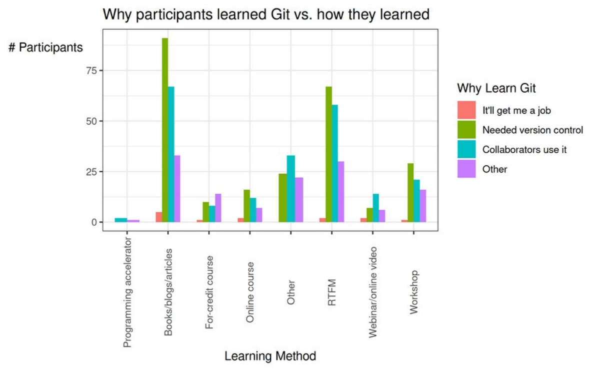

Figure 1. The majority of survey participants had to acquire Git skills through self-teaching methods (e.g. books, blogs).

Likewise, Glassey notes that requiring the use of Git in a classroom setting allows professors to understand student software development patterns and identify bad practices such as waiting to work on projects, having "mega commits", and writing unhelpful commit messages [28]. In one in-depth interview, a professor in digital humanities, explicitly encourages students to work in small chunks and commit, then push often. Through commit messages and tracking minor code changes, the professor is able to understand students' development and decision-making process when creating software, algorithms, or applications (Kadi Part, 2020-06-30_00025). Learning and incorporating Git into one's workflow is "good hygiene" and provides order to an otherwise "chaotic process" of software development (Rocco and Lloyd 2011). Further, Xu notes that Git is a tool with a steep learning curve, but it provides students a competitive advantage and is indicative of working in the real software development environment [29]. Students also reported similar benefits such as gaining relevant experience and project collaboration [30], [31]. Findings from our own focus groups, survey, and interviews align with many of the observations mentioned above. For example, focus group participants were interested in finding a way to "incorporat[e] Git into [their] habit" instead of "procrastinat[ing] on learning really how to use it [... and] wait[ing] for the next project" to pick it up (20200313_transcript_001, 20200410_transcript_002). Together, they provide an interesting student and researcher perspective often missing from behavioral studies, which are written from a computer and information science education perspective (e.g. professorial).

4.4. A Need for Better Opportunities and Solutions

Within these interactive learning opportunities, there are certain aspects of Git and GHP that need to be addressed in order to avoid the pitfalls expressed in the literature and through our behavioral study. The survey data illustrate that the majority of users first used Git as it was incorporated into an existing course or workshop (Figure 1). They also show that those surveyed routinely needed to relearn Git once a semester or even weekly (Figure 2). The diffusion of tools and workflows, and unintuitive terminology, appear to be major factors in steepening the learning curve in order to carry out core concepts of version control with Git, and collaboration on GHP.

In an interview with a mathematics professor, they were confident in speaking about the pull-push concept in order to stay in-sync with collaborators (Interview Question 4), but when explaining why they always use pull origin master terminology to update local and remote repositories they exclaimed: "[...] I don't really know what it does to match with the date, it's still master in this repository anyway […] I hardly know what it does" (Sandra Keskula, 2020-07-09_00042). There was also little to say in explaining Git command syntax—the same math professor couldn't explain the function of a dash but knew they were supposed to type it as part of the command. This aligns with earlier findings of lacking a mental model where a computer science graduate student confuses remote and local repositories, and push/pull being a part of GHP, not as a basic feature of Git itself (Erlend Lippmaa, 20200410_transcript_002). A bioinformatics postdoctoral researcher in a focus group session also noted that they found Git's language structure confusing: ”The whole idea of staging and committing and adding in all these different things and, really, I found [Git] tough to get my head around” (Niki Aghili, 20200710_FocusGroup_003). The sentiments expressed by focus group participants indicate that the opaque command structures and workflow schema are major impediments to learning Git and GHP.

These findings bring to light the trials of learning Git and a clear need for iterative, project-based learning. There is a distinct gap in supporting project-based learning. For example, only 32.8% of the survey participants had a method of onboarding newcomers to version control and coding practices. This is something that faculty, staff, and outgoing students can hold responsibility to provide documentation and onboarding materials for incoming collaborators, a form of peer learning [32]. A bioinformatics Principal Investigator at a public university refers to this as the “knowledge loss problem in academia”, in which labs experience frequent turnover of researchers and capturing and retaining knowledge is a continuous struggle. He is in the process of mitigating this by obtaining “funding that allowed [them] to [...] basically put pipelines online [... and make] it all public, all markdown on GitHub” (Parsia Nazeri, 20200710_transcript_003).

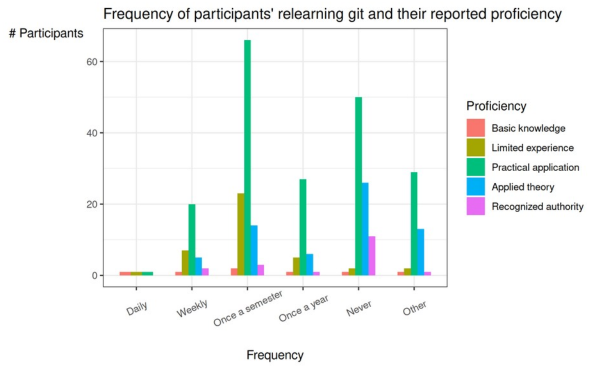

Figure 2. In two separate questions, survey respondents claimed their self-proficiency, as well as their tendency to reteach themselves Git.

Several initiatives exist, however, to scaffold peoples’ understanding and use of Git and GHP. As mentioned, many of the focus group participants attended at least one workshop on Git and GitHub offered by NYU Libraries. While this workshop is offered several times during the semester, there are a number of two-hour workshops available for affiliated students, staff, faculty, and researchers centered on information literacy, new and emerging technologies, and computer programming [33]. This type of course offering is complementary to other efforts to teach Git. Software Carpentry, for instance, has version control with Git as a part of almost every curriculum they teach. These workshops on Git and GitHub can be taught in existing support units, such as libraries, research computing, multidisciplinary centers, and internal department or lab staff. Librarians, in turn, can help bolster existing efforts and also can provide ongoing support for project-based learning with complementary reference and consultation support for Git and GHP.

5. Discussion

Behaviors uncovered from the focus groups, survey, and interviews highlight the significance of Git and GHP as research tools, and also illuminate usability problems that make it difficult to adopt. In spite of these difficulties, however, users were able to see and understand the positive effects of using GHP. For example, in a focus group an engineering master's student, stated that “GitHub is a great tool to introduce the projects you have done. Like some of my friends […] post their GitHub link into their LinkedIn profile. So […] every recruiter can see their results from the projects they have done” (Peyman Meskini, 20200410_FocusGroup_002). Beyond its potential uses in job seeking, participants also noted that using GHP was a way to be involved in a larger conversation. In a similar multiple choice question, we asked survey respondents about the scholarly activities in which they engage on GHP. Respondents reported collaboration, peer production[4], peer review, and publishing scholarship as reflecting the work they currently do on GHP (Figure 3). These answers occurred 490 times out of 993 total responses.

Throughout multiple interviews, participants reiterated that one of three features they would highlight about GHP, in order to convince new users to adopt a particular GHP, was the ease in finding and getting involved in other projects that they wouldn’t have necessarily known about, outside of reading scholarly articles (Interview Question 7). This dovetails with the efforts of Lamprecht et al. and Wilkinson et al. who have taken the 15 FAIR Guiding Principles and modified them to meet the requirements of research software in the hopes of forefronting research software as an important contribution to scholarship that should be preserved [7], [34]. These

4 Peer production is self organization formed around open source projects, and those projects that are built and accessible through GHPs.

revisions consider the complexities of research software—including its distinct needs related to dependencies, interoperability, versioning, and metadata—and adopt, adapt, or reinterpret the FAIR principles in a way that provides a basis for further work in applying them to research software.

Once the software exists, through individual efforts or through collaboration, it is important to preserve those outcomes. One mode of doing this has been through self-archiving. Of the survey participants, 47.2% deposit their code into a repository for long-term storage and archiving. Options for research code include generalist repositories such as Zenodo, Figshare, Dryad, Dataverse, Mendeley Data, and the Open Science Framework. While a copy of the code must be added by the researcher to most generalist repositories, several of these options have integrations with GitHub and GitLab. The GitHub-Zenodo integration, for example, saves a snapshot of the GitHub repository to Zenodo and provides "any software package on GitHub to be given a digital object identifier (DOI), enabling the sharing and preservation of software, and attracting researchers who are familiar with open-source methodology" [35]. Likewise, GitHub accounts can be connected to Figshare and users can enable the Github add-on in the Open Science Framework to create archival copies of their software. Additional options include subject and institutional repositories, both of which have increased support for archiving non-manuscript research outputs, including data, code, and audio visual material. Broadly, supporting self-archiving of software built using Git and GHP provides a more nuanced understanding of the "scholarly Git experience" while also supporting research code's place in the scholarly ecosystem, all of which will likely need much more attention in the future.

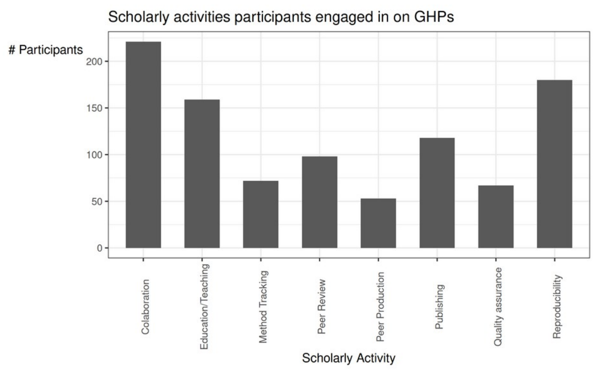

Figure 3. Survey participants identified one or more scholarly activities they engage in on GHP.

This work around self-archiving is promising to see in concert with other software archiving efforts. For instance, Software Heritage has taken up the task of building and designing a universal archive of source code that is focused on "collecting, preserving, and sharing the source code of all the software ever written" [36]. Since 2015, the Software Heritage Foundation has been actively cloning public GitHub repositories via programmatic listers and loaders and now stores copies of code from defunct and active forges such as Google Code, Gitorious, GitHub, GitLab, and many more. Software Heritage's use of intrinsic identifiers (similar to a Git hash) allows citing software objects, including the repository, the file, and even specific lines of code and, therefore, contributing to reproducibility through preservation and enhanced citation for software in research articles [37]. Further, the "Save Code Now" feature in Software Heritage allows users to add their code to the Software Heritage archive in a few short steps.

A final concern in studying and supporting Git in academic environments is how it facilitates scientific reproducibility. Reproducibility is the ability to rerun research workflows with the same data/code and get the same results as the original researcher(s) at any point after publication [38]. This requires not only the original data and code, but also documentation. Such information imparts useful information for reproduction, including software dependencies and workflow. Using GHP can help manage robust versions of projects as they are developed. However, the initial data from our focus groups show that these benefits may be lost as a result of the difficulty in using Git. Focus group participants admitted, for example, that Git's unintuitive language made it unclear on how to integrate "good project management [...] like when to branch, or when to fork" (Niki Aghili, 20200710_FocusGroup_003). Many interview participants further conflated cloning and forking when asked to edit multiple files from a Git repository that they did not have contributor access to (Interview Scenario 6).

This highlights the need for more communal best practices to get projects started on the right foot, as well as the infrastructure to be able to interact with code and scholarly ephemera to understand it in the long-term. The Software Preservation Network (SPN) has been an impactful contributor when it comes to raising awareness regarding software, and has made some major contributions to the software preservation landscape [39]. SPN affiliated projects—such as Best Practices in Fair Use for Software Preservation, Fostering Communities of Practice: Software Preservation and Emulation in Libraries, Archives and Museums (FCoP), and Emulation-as-a-Service Infrastructure (EaaSI)—have contributed not only to the technological and legal infrastructures needed for software preservation and reuse, but also helped to formalize software preservation and software emulation roles in GLAM sectors. These awareness-raising efforts are important elements in helping the scholarly community reconceptualize how we think about source code and how to structure and document that work towards fulfilling the need to capture and preserve it so that it is accessible and rerunnable for present-day and future researchers.

6. Conclusion

Our findings from the IASGE behavioral study reveal interesting patterns in Git and GHP usage in academia. By centering our research question on behaviors related to learning and adopting Git, and providing context for the focus groups, survey, and interviews, we have begun the conversation of how and why VCS are used for scholarly research. In this paper, we have presented preliminary findings that explicitly express the experiences and behaviors of both new and seasoned Git users. These include the frustrations related to learning and adopting Git, as well as the benefits of VCS as a collaborative tool. Future directions for this project include additional focus groups, full qualitative analysis of the behaviors witnessed during the participatory user sessions, educational material for librarians and scholars related to teaching and onboarding Git and GHP, and partnerships with allied projects. With these goals in mind, there is great potential in supporting Git usage in academia and hope that others will begin to value source code and its contributions to the scholarly record.

7. Acknowledgments

The authors contributed equally to this paper and are listed in alphabetical order. The authors wish to thank the Alfred P. Sloan Foundation for their generous funding of this project, especially our program officer Josh Greenberg. We would also like to thank the people that generously provided feedback on the initial draft of this paper: Rémi Rampin and wilkie.

8. Supplementary Files

The data, documentation, and source code underlying our survey analyses that were done for HICSS can be found here: https://osf.io/57tb8/. Our qualitative data and associated documentation can be found here: https://doi.org/10.5064/F6VOIB8H.

9. References

[1] M. Squire, "The Lives and Deaths of Open Source Code Forges," in *Proceedings of the 13th International Symposium on Open Collaboration - OpenSym '17*, Galway, Ireland, 2017, pp. 1–8, doi: 10.1145/3125433.3125468.
[2] D. Chan, "Sunsetting Mercurial support in Bitbucket," *Bitbucket*, Apr. 21, 2020. https://bitbucket.org/blog/sunsetting-mercurial-support-in-bitbucket (accessed Jul. 02, 2020).
[3] A. Favell, "The History of Git: The Road to Domination," *Welcome to the Jungle*, Feb. 04, 2020. https://www.welcometothejungle.com/en/articles/btc-history-git (accessed May 27, 2020).
[4] L. Torvald, "'Kernel SCM saga..' - MARC," *linux-kernel*, Apr. 06, 2005. https://marc.info/?l=linux-kernel&m=111280216717070&w=2 (accessed Jul. 06, 2020).
[5] StackOverflow, "Stack Overflow Developer Survey 2018," *Stack Overflow*. https://insights.stackoverflow.com/survey/2018/ (accessed Jul. 15, 2020).
[6] M. Stebbins, "Expanding Public Access to the Results of Federally Funded Research," *whitehouse.gov*, Feb. 22, 2013. https://obamawhitehouse.archives.gov/blog/2013/02/22/expanding-public-access-results-federally-funded-research (accessed Jul. 02, 2020).
[7] M. D. Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Sci Data*, vol. 3, no. 1, pp. 1–9, Mar. 2016, doi: 10.1038/sdata.2016.18.
[8] K. B. Read, L. Amos, L. M. Federer, A. Logan, T. S. Plutchak, and K. G. Akers, "Practicing what we preach: developing a data sharing policy for the Journal of the Medical Library Association," *Journal of the Medical Library Association*, vol. 106, no. 2, Art. no. 2, Apr. 2018, doi: 10.5195/jmla.2018.431.
[9] L. R. Johnston, "Barriers to Collaboration: Lessons Learned from the Data Curation Network," *Research Library Issues*, no. 296, pp. 37–43, Dec. 2018, doi: 10.29242/rli.296.5.
[10] L. M. Kellam and K. Thompson, Eds., *Databrarianship: the academic data librarian in theory and practice*. Chicago: Association of College and Research Libraries, a division of the American Library Association, 2016.
[11] M. J. Sevigny, "A descriptive study of instructional interaction and performance appraisal in a university studio art setting : a multiple perspective /," The Ohio State University, 1977.
[12] H. F. Wolcott, "Ethnographic Research in Education," in *Complementary Methods for Research in Education*, Washington, D. C.: American Educational Research Association, 1988, pp. 185–206.
[13] R. Glassey, "Adopting Git/Github Within Teaching: A Survey of Tool Support," in *Proceedings of the ACM Conference on Global Computing Education*, New York, NY, USA, 2019, pp. 143–149, doi: 10.1145/3300115.3309518.
[14] A. J. Onwuegbuzie, W. B. Dickinson, N. L. Leech, and A. G. Zoran, "A Qualitative Framework for Collecting and Analyzing Data in Focus Group Research," *International Journal of Qualitative Methods*, vol. 8, no. 3, pp. 1–21, Sep.

2009, doi: 10.1177/160940690900800301.
[15] R. Rampin, *Taguette*. 2020.
[16] S. Cowan and J. McLeod, "Research methods: Discourse analysis," *Counselling and Psychotherapy Research*, vol. 4, no. 1, pp. 102–102, 2004, doi: 10.1080/14733140412331384108.
[17] A. J. Pickard, *Research methods in information*. London: Facet, 2007.
[18] C. A. Scherbaum and K. M. Shockley, *Analysing Quantitative Data for Business and Management Students*. 1 Oliver's Yard, 55 City Road London EC1Y 1SP: SAGE Publications, Inc., 2015.
[19] B. Friedman, P. H. Kahn, and A. Borning, "Value Sensitive Design and Information Systems," in *Human-computer Interaction and Management Information Systems: Foundations*, Routledge, 2006.
[20] J. Bryan, "Excuse Me, Do You Have a Moment to Talk About Version Control?," *The American Statistician*, vol. 72, no. 1, pp. 20–27, Jan. 2018, doi: 10.1080/00031305.2017.1399928.
[21] C. Bacharakis, "Cracking the Code — how Mozilla is helping university students contribute to Open Source," *Medium*, Jun. 26, 2018. https://medium.com/mozilla-open-innovation/cracking-the-code-how-mozilla-is-helping-university-students-contribute-to-open-source-25fa630d8c5c (accessed Jul. 15, 2020).
[22] V. Isomöttönen and M. Cochez, "Challenges and Confusions in Learning Version Control with Git," in *Information and Communication Technologies in Education, Research, and Industrial Applications*, Nov. 2014, vol. 469, pp. 178–193, doi: 10.1007/978-3-319-13206-8_9.
[23] M. Schwern, "[Linux.conf.au 2013] - Git For Ages 4 And Up," presented at the Conference of Australian Linux Users talk, Australia, Mar. 25, 2013, Accessed: Jul. 08, 2020. [Online]. Available: https://www.youtube.com/watch?v=1ffBJ4sVUb4&feature=emb_title.
[24] "diff," *IEEE, The Open Group Base Specifications Issue*, 2018. https://pubs.opengroup.org/onlinepubs/9699919799/utilities/diff.html (accessed Oct. 03, 2020).
[25] "diff page from Section 1 of the inferno manual," *Manual Page archive*. http://man.cat-v.org/inferno/1/diff (accessed Oct. 03, 2020).
[26] "diff page from Section 1 of the plan 9 manual," *Manual Page archive*. http://man.cat-v.org/plan_9/1/diff (accessed Oct. 03, 2020).
[27] A. Athalye, J. Javier, and J. Gjengset, "Why we are teaching this class," *the missing semester of your cs education*. https://missing.csail.mit.edu/about/ (accessed Jul. 15, 2020).
[28] L. Glassy, "Using Version Control to Observe Student Software Development Processes," *J. Comput. Sci. Coll.*, vol. 21, no. 3, pp. 99–106, 2006.
[29] Z. Xu, "Using Git to Manage Capstone Software Projects," in *ICCGI 2012: The Seventh International Multi-Conference on Computing in the Global Information Technology*, 2012, pp. 159–164.
[30] J. Feliciano, M.-A. Storey, and A. Zagalsky, "Student Experiences Using GitHub in Software Engineering Courses: A Case Study," in *Proceedings of the 38th International Conference on Software Engineering Companion*, New York, NY, USA, 2016, pp. 422–431, doi: 10.1145/2889160.2889195.
[31] L. Haaranen and T. Lehtinen, "Teaching Git on the Side: Version Control System As a Course Platform," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA, 2015, pp. 87–92, doi: 10.1145/2729094.2742608.
[32] J. Fehr, C. Himpe, S. Rave, and J. Saak, "Sustainable Research Software Hand-Over," *arXiv:1909.09469 [cs]*, Sep. 2019, Accessed: Sep. 30, 2019. [Online]. Available: http://arxiv.org/abs/1909.09469.
[33] S. Guss, "A Studio Model for Academic Data Services," in *Databrarianship: The Academic Data Librarian in Theory and Practice*, L. M. Kellam and K. Thompson, Eds. Chicago: Association of College and Research Libraries Press, 2016, pp. 9–24.
[34] A.-L. Lamprecht *et al.*, "Towards FAIR principles for research software," *Data Science*, vol. Preprint, no. Preprint, pp. 1–23, 2019, doi: 10.3233/DS-190026.
[35] GitHub, "Making Your Code Citable · GitHub Guides," 2016. https://guides.github.com/activities/citable-code/ (accessed Jul. 11, 2019).
[36] R. Di Cosmo, "Software heritage: collecting, preserving, and sharing all our source code (keynote)," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Montpellier, France, Sep. 2018, pp. 1–2, doi: 10.1145/3238147.3241985.
[37] R. Di Cosmo, "Saving and referencing research software in Software Heritage," *Software Heritage*, Aug. 05, 2019. https://www.softwareheritage.org/2019/08/05/saving-and-referencing-research-software-in-software-heritage/ (accessed Sep. 23, 2019).
[38] K. Bollen, J. Cacioppo, R. Kaplan, J. Krosnick, and J. Olds, "Reproducability, Replicability, and Generalization in the Social, Behavioral, and Economic Sciences," 2015.
[39] J. Meyerson *et al.*, "The Software Preservation Network (SPN): A community effort to ensure long term access to digital cultural heritage," *D-Lib Magazine*, vol. 23, no. 5–6, p. 1p., 2017, doi: 10.1045/may2017-meyerson.