Understanding Git History: A Multi-Sense View

Kevin J. North
Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE, 68588-0115 USA
knorth@cse.unl.edu

Anita Sarma
Electrical Engineering & Computer Science
Oregon State University
Corvallis, OR, 97331 USA
anita.sarma@oregonstate.edu

Myra B. Cohen
Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE, 68588-0115 USA
myra@cse.unl.edu

ABSTRACT

Version control systems archive data about the development history of a project, which can be used to analyze and understand different facets of a software project. The project history can be used to evaluate the development process of a team, as an aid in bug fixing, or to help new members get on track with development. However, state of the art techniques for analyzing version control data provide only partial views into this information, and lack an easy way to present all the dimensions of the data. In this paper we present GitVS, a hybrid view that incorporates visualization and sonification to represent the multiple dimensions of version control data - development time line, conflicts, etc. In a formative user study comparing the GitHub Network Graph, GitVS, and a version of GitVS without sound, we show GitVS improves over the GitHub Network Graph and that while sound makes it easier to correctly understand version history for some tasks, it is more difficult for others.

CCS Concepts

•Software and its engineering → Software configuration management and version control systems;

Keywords

Version Control History, Conflicts, Sonification

1. INTRODUCTION

Software teams use version control to manage their projects. As a result, version control systems record a rich project history containing information about who made commits, how many files the commits modify, when branches are forked or merged, and in some cases when conflicts occur. Developers and managers may study patterns of information contained in this history to understand why a piece of code evolved, to keep up with changes to pieces of code which interest them, or to find and debug faults [4].

SSE'16, November 14, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4397-8/16/11...$15.00
http://dx.doi.org/10.1145/2993283.2993285

Several techniques exist for displaying version control history. One common technique is to use a directed graph with commits adjacent to their predecessors. The GitHub network graph is an example of such a visualization [15]. Another technique, code_swarm [14], shows relationships between developers and their edits to files as a video. In recent work, we developed GitSonifier, a technique that uses audio sounds to represent conflicts and developers while differentiating days along a timeline [13]. Each technique has strengths, yet each has limited dimensions of information. For instance, the GitHub network graph and code_swarm do not provide conflict information, while GitSonifier does not contain file information, making it hard to target a single point in time.

In this paper, we take a step back and ask how can we present more information at once while continuing to allow users to easily understand the version history information. We focus on a multi-sense approach that utilizes sonification, the use of sound to portray data. We have created *GitVS*, a hybrid visualization for version control history that incorporates multiple dimensions of version control data visually, combined with a *sonification cursor* UI element that allows one to listen to who made each commit and where historical conflicts occurred. We perform a formative user study to evaluate GitVS with the GitHub Network Graph, a popular Git history visualization. To evaluate the effect of sonification on usability we created a variation of GitVS with no sound (GitVS-NS) and use it as another treatment in the user study. In a between-groups study with three participants per treatment, we find that GitVS outperforms the GitHub network graph in helping users correctly and efficiently understand version history. Moreover, its portrayal of developers using sound is more helpful than finding the same information without sound, but its other uses of sound can be improved.

The contributions of this work are: (1) a novel technique, GitVS, for visualizing and understanding version control history from multiple dimensions, (2) a user study that shows GitVS improves over the GitHub Network Graph, and (3) identification of which tasks are helped by sound when investigating the version control history and which are not.

2. BACKGROUND AND RELATED WORK

2.1 GitSonifier

Sonification is the portrayal of data using sound. Sonification is effective at portraying multidimensional data because sound has multiple characteristics, including frequency, loud-

ness, and rhythm, that can be mapped to different pieces of data. In addition, sonification is well-suited to portray data that is temporal in nature. [9]

In GitSonifier (and GitVS) we use two specific types of sonification: Earcon sonification and parameter mapping sonification (PMSon). In earcon sonification, each piece of data is represented by a different short phrase of music called an earcon. Earcons are typically differentiated by using different notes, rhythms, and instruments. In PMSon, the properties of a sound wave are mapped to different aspects of the data being portrayed. For example, to sonify the temperature of a pot of boiling water, the temperature could be mapped to a soundwave's pitch. As water is heated, the sonification rises in pitch. [9]

GitSonifier represents three pieces of data from version control history: (1) who makes each commit, (2) when each commit is made, and (3) when conflicts are introduced and resolved in the project history. Music is produced along the version history timeline. Each *measure* represents one commit. A measure is a basic unit of rhythm in music theory, and untrained listeners can identify measures intuitively [19]. When a commit's measure is played, an earcon corresponding to the developer who made that commit is played. The earcons use different rhythms, notes, and instruments in order to make them easily distinguishable.

To show time passing, we use an earcon called a day separator. Whenever one calendar day ends, we insert a measure with the day separator. If multiple days pass without commits, we play multiple consecutive day separators to count the number of days with no activity. To distinguish the day separator from developer earcons, we use a different musical key and a distinctive synthesizer instrument.

We play drums in the background to indicate when conflicts are introduced and resolved. When a commit adds a conflict, drums begin playing during that commit's measure. When a conflict is resolved, the drums stop. If a second conflict appears before an existing conflict is resolved, the drums become louder (using PMSon). When either conflict is resolved, the drums return to the previous volume.

In a formative user study we found that participants could easily answer questions about version history using GitSonifier. However, our study also revealed some weaknesses. Because it uses only sound, one cannot jump to a different point in time in history with GitSonifier, making exploring the data on a specific date challenging. In addition, while participants understood specific pieces of data, it was more difficult for them to understand the history holistically. [13]

2.2 Related Work

There has been some work on visualizing project history. For example, Syde [8] presents which developers have changed which files and how code ownership changes on a file as the project evolves. Workspace Activity Viewer [17] replays version history regarding which files have changed and by how much over time as a 3D animation. Scamp [11] portrays how much a file has changed using heat maps and how often classes have been edited using a word cloud. None of these visualizations allow drill-down investigation of specific events or provide conflict information.

Workspace awareness tools attempt to proactively identify conflicts as they emerge via workspace monitoring (e.g., Palantir [18], FastDash [10]) or by merging changes in a shadow repository (e.g., Crystal [3], SafeCommit [22]). While these tools present conflict information, they only show conflicts that are happening at the present time. None of these tools provide a historical perspective of conflicts.

McIntosh et. al. introduced a method to systematically generate music based on version control histories. Unlike GitVS, this technique is solely intended to be enjoyable to listen to, not to aid understanding version history [12].

CocoViz [2] is the closest approach to GitVS. It combines visualization and sonification to represent code metrics for source code files. Each file is represented by a shape, and properties of a shape (e.g., size and color) are mapped to code metrics. Sound represents additional code metrics when the user hovers over a shape. CocoViz does not use or present information about version control history, however.

3. GITVS

3.1 Visualization and Sonification Design

GitVS combines the GitSonifier sonification with a two-dimensional visualization. It shows each commit's timestamp, ID, committer, and list of modified files, as well as the project's branching and merging patterns. It also shows when conflicts were introduced and resolved in the project's history.

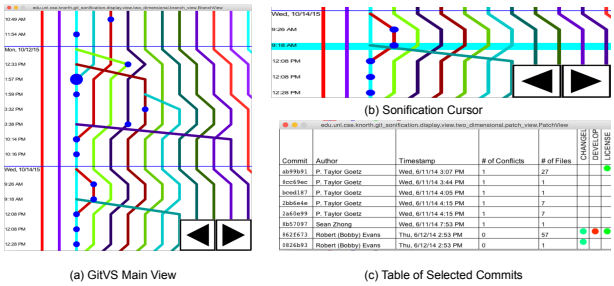

Figure 1: GitVS Views: (a) main window, (b) sonification Cursor, (c) "details view" of selected commits

Our implementation works with Git repositories, but the technique can be applied to other version control systems. Figure 1 shows three views of GitVS. Figure 1(a) shows the main screen. The vertical axis represents time moving from top to bottom. The user can scroll and zoom to see more of the timeline providing scalability. Commits are shown as circles, and each commit timestamp is shown in the left margin. The more files a commit modifies, the larger is the commit circle. Horizontal bars indicate when each calendar day begins and ends; days without any commits are skipped. Each branch is drawn in a different color.

Figure 1(b) shows the sonification cursor, the horizontal blue bar. The user can click the large arrows on the screen to move the cursor up and down, or right-click on a commit which moves the sonification cursor to that location. When the cursor touches a commit's circle, a developer earcon plays for that commit. Day separators play between calendar days, and are played multiple times consecutively for skipped days. Drums sounds play for conflicts using the same sonification design as GitSonifier. In our implementation, the 13 most prolific developers in the entire repository each get their own individual developer earcon sound, and the remaining developers share a 14th sound. (This number

represented developers who made 75% of the commits in the repository for our first user study [13].)

Finally, GitVS lets users click and drag over one or more commits to get additional details about these commits via the "details view" (Figure 1(c)). This view shows the selected commits' IDs, timestamps, contributors, and number and the names of files modified.

We also created GitVS-No Sound (GitVS-NS), a tool that modifies GitVS by removing all sounds associated with the development history (and the sonification bar). In the main view, a red border on the commit circle represents commits with at least one conflict. When commits are selected by a user, an additional column that shows the exact number of conflicts present during a commit is shown in the "details view". While GitVS-NS uses different techniques to show data, it provides access to exactly the same pieces of information as GitVS.

3.2 Implementation Details

GitVS takes the following information as input: (1) the version control repository to analyze and display, (2) the first and last commits to display in the view, and (3) a list of historical conflicts in the repository (this is obtained via a separate tool that processes Git data). Item #2 allows users to specify a date range. For item #3, we use a Python script to extract the conflict information that is then sent to GitVS. We provide a virtual machine with GitVS that has instructions on how to automate the collection of conflict information for repositories.[1]

GitVS goes through the repository and, for all of the commits, collects each commit's committer, timestamp, SHA1 hash, list of parent commits, list of files modified, and position within the project's branching and merging structure. Next, it assigns each developer an earcon based on the number of their commits, filters out commits that don't fit in the date range specified, and determines where day separators should be inserted. Finally, GitVS produces the visualization and sonification. GitVS was implemented in Java 1.7. It uses jGit [20] to interact with Git repositories and Processing [16] to implement the interactive user interface.

4. USER STUDY

We conducted a formative user study to investigate how developers can use GitVS to understand version history; how it compares to a popular existing tool, the GitHub network graph [15]; and how sound helps in understanding version history.

In our study, we asked the following research questions:

- RQ1: How does GitVS compare to the GitHub Network Graph and GitVS-NS in terms of: (a) effectiveness, (b) efficiency, and (c) users' opinion of the tool?
- RQ2: Which factors do participants consider when analyzing version history, and how does the usage of a particular tool affect their focus?

To answer these questions, we used a between-groups study with three treatments. One group used the GitHub network graph (Figure 2(a)), a tool from GitHub and frequently used by developers. The second group used a version of GitVS with No Sound (Figure 2(b)), and the third group used GitVS (Figure 2(c)), which allows us understand the role of sonification in users' experiences.

There were three parts of the user study. First, participants were trained on the tool that they were going to use, after which they performed the study. We performed an exit survey and interview at the end.

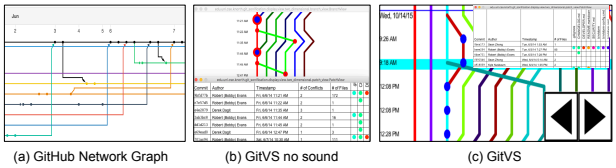

Figure 2: Treatments used in study: (a) GitHub Network Graph (b) GitVS-NS (c) GitVS

4.1 Participant Demographics

We recruited nine computer science students: six undergraduate and three graduate students. All participants had at least one year of experience with Git. The most experienced participant had over five years of experience. Many had used other other version control systems (e.g., SVN, CVS, Mercurial, Bazaar). All participants had used version control on at least one professional or major class project. Three participants had eight months to 1.5 years of experience with continuous integration. Two participants had musical experience. One participant was female, and the rest were male. The demographics of the participants were collected before they were assigned to a group in a random fashion. By chance the two participants with music experience were split evenly between GitVS and GitVS-NS.

4.2 Study Scenario

Participants were asked to take on the role of a manager of a development team and play out the following scenario:

The team currently uses the following development processes: (1) Whenever a new feature is being implemented, a developer can create a new branch but is not required to, and (2) Branches are code-reviewed before being merged. The team, however, is facing disruption to its workflow because of conflicts and the manger is considering making the use of a feature branch a requirement and to have the team use a continuous integration server. Before finalizing this requirement the manger wants to compare the two processes. So he/she investigates his/her own project and another similar project that uses the new processes.

We used two open source projects for this scenario. Voldemort [6] is a distributed storage system, represents the manager's current process. Storm [5] is a distributed computation system and represents a project with the new processes. Both projects are fairly large. Voldemort has over 4,100 commits and started in April 2011. Storm has over 6,500 commits and started in September 2011. Both have a similar amount of developer activity.

4.3 Study Design

The experiment was run in three steps. First, participants were trained on dummy data. Second, they performed the experiment tasks, after which they completed an exit survey. Last, we interviewed them about specific aspects of their task or decision. They were then paid $20.

[1] http://cse.unl.edu/~myra/artifacts/GitVS/vm

Training Task: Participants were shown a video demonstrating the use of the assigned tool. This video uses version history from a fake project with 28 commits, 6 branches, 4 developers, and 3 files lasting 8 days.

Once participants had seen the video and played with the tool, they were asked six questions about the data portrayed by the tool. A participant could move to the experiment tasks only after he or she correctly answered all the training questions. When answering questions, participants were able to review the training video and use the tool. No data was collected during this phase.[2]

Study Tasks: There were three phases in the experiment. In Phases I and II, participants investigated projects that either followed the current or *new* processes. The order in which a participant viewed a repository was assigned at random to a phase and was then counter balanced. For each phase, participants were asked a set of objective questions:

1. How many commits were made in-between [three of the days portrayed by the tool]?
2. On average, what is the number of files modified per commit in-between [three of the days portrayed by the tool]?
3. How many conflicts were present in-between [the first and last date portrayed by the tool]?
4. Look at the branches which were merged directly into the master branch. In days, what is the average number of days these branches existed?
5. What is the average number of days each branch with at least one commit from an outsider developer existed before being merged?

In Phase III, participants had to compare the two projects to answer the following questions. When answering each question, participants had to select one of the projects and provide a justification. During this phase participants could review their answers to Phases I & II.

1. Based on the previous questions, which process would you recommend if your priority is to have a large number of small commits?
2. Based on the previous questions, which process would you recommend if your priority is to minimize conflicts?
3. Based on the previous questions, which process would you recommend if your priority is to accept pull requests quickly?
4. Based on the previous questions, which process would you recommend if your priority is to quickly merge commits from outside developers?

These questions were selected because they reflect methodologies that affect team efficiency and therefore are of interest to managers. Q #1 is based on industry best practices that small frequent commits make sharing changes easy [1]. Q #2 reflects findings about conflicts adversely affecting teams [7]. Q #3 and #4 reflect research that shows that teams using continuous integration merge more pull requests from core developers and reject fewer pull requests from non-core developers [21].

During the experiment we only answered questions from participants when they pertained to correctly interpreting the questions. We did not answer questions about how to use the assigned tool or explain the data. If a participant took longer than five minutes to answer a question, we told them that they could skip the question if they wished.

Data used in Study: We used a two week-long portion of history from each project. The data selected from Storm had more commits (77 vs. 40), branches (11 vs. 6), and developers (107 vs. 34 in the entire repositories, 7 vs. 4 in the two weeks shown) than the data from Voldemort.

In addition, participants were given lists of core developers and non-core developers for each repository.[3] Participants who used GitVS could also listen to developer earcons in the lists. Since the training, original, and comparison repositories had different developers, and GitVS uses the same set of (developer) earcons for each repository, the developers associated with a particular earcon was specific to a repository.

Post Study Tasks: At the end of the study, participants filled out a survey asking opinions about using the tool. These 10 questions (listed in Table 3) used a Likert scale from 1 (least likely) to 5 (most likely).

Three of the questions also include a "(Why or why not?)" option, for which participants had to provide a short explanation for their Likert rating. There were 2 additional open-ended questions:

- Would you recommend this tool to others?
- In your opinion, how can we improve the tool further?

Finally, we conducted an exit interview, which were audio recorded. These were unstructured interviews, allowing us to ask participants about any specific activity we thought was interesting or unusual.

5. RESULTS

5.1 RQ1: Comparing GitVS to the GitHub Network Graph and GitVS-NS

5.1.1 Effectiveness

Table 1 shows how many questions participants answered correctly in Phases I & II (Table 2 provides a summary). Three participants used each tool, so each question can have up to three correct answers. GitVS and GitVS-NS participants gave 16 correct responses per tool out of 30 questions. In contrast, only one correct response was recorded from GitHub participants. There are several possible reasons for why participants using GitHub answered questions incorrectly. For Q1 and Q2, the dots representing commits are small and there is no zoom feature, so we think it was difficult to keep track of which commits had already been counted. For Q3, GitHub does not record conflict data. As a result, participants needed to manually look through each commit's list of changes to see if it modified the same file in the same place as another commit on another branch, a very time-consuming process.

For Q4 and Q5, GitHub participants frequently miscounted the number of branches. We think this is because the spacing between branches is inconsistent and there is no way to annotate branches to keep track of which have been counted. The GitHub network graph also has a bug that incorrectly labeled some commits with a timestamp a day too early in the main visualization. The correct date is shown when

[2]Study questions and results are on our website: http://cse.unl.edu/~myra/artifacts/GitVS/

[3]Core developers are the developers who have contributed to about 75% of the project's commits; the rest are non-core.

Table 1: Data from the objective questions.

Repository	Question	Tool	Correct Answers (out of 3)	Avg. time to answer (minutes)
Voldemort	Q1	GitHub	0	0.65
		GitVS-NS	2	0.91
		GitVS	3	1.34
Voldemort	Q2	GitHub	0	3.00
		GitVS-NS	3	1.86
		GitVS	3	2.72
Voldemort	Q3	GitHub	1	2.08
		GitVS-NS	2	1.07
		GitVS	2	2.30
Voldemort	Q4	GitHub	0	2.16
		GitVS-NS	1	2.01
		GitVS	2	1.66
Voldemort	Q5	GitHub	0	2.07
		GitVS-NS	2	3.25
		GitVS	1	3.92
Storm	Q1	GitHub	0	1.07
		GitVS-NS	2	1.46
		GitVS	2	1.28
Storm	Q2	GitHub	0	2.04
		GitVS-NS	2	2.13
		GitVS	2	1.48
Storm	Q3	GitHub	0	2.72
		GitVS-NS	1	2.64
		GitVS	1	5.95
Storm	Q4	GitHub	0	3.20
		GitVS-NS	1	4.74
		GitVS	0	4.83
Storm	Q5	GitHub	0	3.50
		GitVS-NS	0	5.66
		GitVS	0	4.82

Table 2: Overall results for objective questions.

Tool	Correct Responses (out of 30)	Average time to answer (minutes)	Std. dev. time to answer (minutes)
GitHub	1	2.69	2.03
GitVS-NS	16	2.25	1.04
GitVS	16	3.03	2.13

clicking on a commit for more details. This bug did not affect the answers to any of the questions in Storm repository, but it did affect Q1, Q2, Q3, and Q5 in Voldemort. We considered answers to be correct if participants used the faulty dates from the visualization or the correct dates from the actual commits. Only one participant gave an answer that could be marked correct either way.

For Q2 and Q3 for Voldemort and Q1, Q2, and Q3 for Storm, we received the same number of correct responses from GitVS and GitVS-NS participants. For Q1 and Q4 in Voldemort, we received one more correct response from GitVS participants than GitVS-NS participants. The GitVS-NS participant who got Q1 incorrect, P8, said that he thought the question "*was a big question and there were a lot of intricacies to how that question had to be answered*," For Q4, most participants for both GitVS and GitVS-NS who answered incorrectly did not notice that the visualization skipped days when there was no activity, leading them to miscalculate the length of branches. While day separators in the GitVS sonification were meant to help participants avoid this mistake, participants usually opted to cut the day separators sound off early and focus on other sounds.

In Q5 for Voldemort and Q4 for Storm, we received one more incorrect response from GitVS participants (as compared to GitVS-NS). For Q5 for Voldemort, the GitVS participants who answered incorrectly did not realize that days (without activity) were omitted. The GitVS-NS participant who answered the question incorrectly, P8, only found the duration of one branch before submitting their answer to avoid taking a long time. No participants for either tool answered Q5 for Storm correctly.

Our results suggest that GitVS is more effective in helping users understand Git data than the GitHub Network Graph. Further, sound neither hurt nor helped participants to understand the data correctly. Overall, participants performed slightly better in the Voldemort project as it was less complex than Storm.

5.1.2 *Efficiency*

Table 1 shows the average time (in minutes) that participants took to answer each question in Phases I & II. Most participants using GitVS took approximately as long or longer than GitHub participants to answer individual questions. Averaging across all questions and both projects show participants using GitHub took 2.69 minute as compared to 3.03 for GitVS, as shown in Table 2. However, since GitHub participants answered nearly all questions incorrectly, it is not clear whether GitHub is faster because it is more efficient or it is so obtuse that participants gave up trying to find correct answers. Another possibility is that GitHub encouraged incorrect investigation patterns that are faster but produce incorrect results.

For several questions, such as Q1, Q2, Q3, and Q5 for Voldemort and Q3 for Storm, GitVS participants took longer to answer questions than GitVS-NS participants. However, GitVS participants answered Q4 of Voldemort and Q1, Q2, and Q5 of Storm more quickly. We think this may be due to learning effects. Two of the GitVS participants completed the Voldemort repository first, but two of the GitVS-NS participants completed Storm first. Hence, the GitVS participants were likely to complete Voldemort, the first repository they saw, more slowly than Storm, and visa-versa for GitVS-NS.

As shown in Table 2, overall, GitVS-NS participants answered questions more quickly (2.25 min) than GitVS participants (3.03 min). One participant for GitVS did not realize that he could scroll through the GitVS visualization, leading him to take much longer than other participants. Given our small sample size, this highly skewed the results.

5.1.3 *Participants' Opinions*

Table 3 shows the average scores of the post-study survey. These use a Likert scale from 1 to 5, 5 being the most positive. For most of the questions, GitVS has an average score 0.67 to 1 point higher than GitHub. However, for the last three questions, GitHub has scores that were much higher.

Q1 through Q6 ask about how easy it is to understand specific data points. The wording of Q7–Q10, including the questions where GitHub scored higher, focus on whether the tool makes it easier to understand the data holistically and whether the participant would share the tool with others. Our results indicate that participants preferred GitVS for finding specific details, but found GitHub to be better for understanding the big picture.

Table 3: Average scores from post-study survey.

Question #	Question	Tool	Avg. Score
Q1	It was easy to tell when each commit was made.	GitHub	4
		GitVS-NS	5
		GitVS	5
Q2	It was easy to identify who made each commit.	GitHub	5
		GitVS-NS	3.33
		GitVS	4.33
Q3	It was easy to tell where conflicts were located.	GitHub	2.33
		GitVS-NS	3.67
		GitVS	3
Q4	It was easy to tell when each commit was made.	GitHub	4.33
		GitVS-NS	5
		GitVS	5
Q5	It was easy to see which files were changed by each commit.	GitHub	2.33
		GitVS-NS	4
		GitVS	3
Q6	It was easy to tell where branches were located.	GitHub	4
		GitVS-NS	4
		GitVS	3.33
Q7	It was easy to understand the details of the development data.	GitHub	2.67
		GitVS-NS	3.67
		GitVS	3.33
Q8	It was easy to understand overall patterns in the development data.	GitHub	3.67
		GitVS-NS	3.33
		GitVS	2.33
Q9	It easy to see relationships between different pieces of development data.	GitHub	3.33
		GitVS-NS	3.33
		GitVS	2.67
Q10	I would be interested in using this tool for my own team.	GitHub	4
		GitVS-NS	4
		GitVS	2.67

In the opened-ended questions, we received mixed responses. Participants disagreed on whether GitHub visualizations were helpful. For example, P2 wrote, "*...because of the branch view, it is easy to see the structures*," but P6 said, "*There's no clear path seen in the network view, especially when there's many branches.*"

GitVS participants also had mixed feelings about the usefulness of sound. For example, P3 wrote that he appreciated GitVS' "*use of colors, text, music rather than just text*," but P5 wrote, "*I was so bogged down with trying to compare sounds in my head that I completely lost sight of what I was actually seeing & trying to find out.*"

In general, participants who used GitVS-NS had higher opinions of their tool than GitVS participants. For Q1 and Q4, both tools received the same average score, and for Q2, GitVS actually scored higher.

For the other seven questions, GitVS-NS received higher scores. The question on which GitVS outperformed GitVS-NS asks, "it was easy to identify who made each commit." We believe that the sonification made it easier to discover commits' developers. In GitVS-NS, a user has to select the commit to open the details view to find the information. In the comments some participants of GitVS-NS said that they would like to have the names of the developers available on the main visualization. P8 wrote, "*It was grueling to figure out which branches were committed to by at least 1 [non core] developer. Names weren't on the graph.*" In contrast, in GitVS, one can listen to a commit's sound directly. Nonetheless, GitVS with sound scored more poorly on other questions, including questions that do not have anything to do with sonification. This suggests that sonification may distract from understanding individual pieces of data that aren't part of the sonification. We need to perform further study to tease out the effects of sonification.

GitVS scored lower than GitVS-NS for Q3, even though conflicts are sonified in GitVS. We believe that GitVS scored lower because in order to find when a conflict occurs the user has to listen to many commits and compare drum sounds between them. This is a multi-step process, as compared to the developer earcons, which allow a user to simply click on a commit and immediately hear who made the change.

For Q10, GitVS scored only 2.67 on average while GitHub and GitVS-NS scored 4, more than one full point higher. In their comments, all of the GitHub and GitVS-NS participants enthusiastically said they would recommend the tool, writing comments such as, "*Yes, it would be better to have a tool to track all the changes and it would be easier to code review*" (P2). In contrast, GitVS participants specifically said they were concerned that the use of sound would make it difficult to share the tool with others. For example, P1 wrote, "*If I know someone more musically inclined then myself maybe. I think this would be helpful for someone who is a very auditory person.*" P5, who had musical experience, said that she would not recommend the tool to others, but did not explain her decision.

The questionnaire results show that participants prefer sonification when it allows them to obtain more information with fewer steps, but when finding the same information requires multiple steps, the sonification can be outperformed by a visualization. This suggests that we need to redesign the sonification to be simpler.

5.2 RQ2: Which factors do participants consider when analyzing version history?

In Phase-III participants had to select one of the two projects (Voldemort or Storm) when answering the four subjective questions. Voldemort was selected 4 to 5 times, and Storm, 7 to 8 times, irrespective of the treatment (tool). We did not find any correlation to suggest that the tool affected the selection when answering Phase-III questions.

Participants also explained the reasoning for their (project) selection in Phase-III. We analyzed their responses to discover which types of data was used by participants in each treatment group. Figure 3 shows how often participants referenced several factors in version history data. Since there were three participants and four question explanations, each factor could have appeared up to 12 times per tool.

GitVS-NS participants mentioned the number and length of branches more frequently than GitVS participants, but mentioned the organization of branches and workflow less frequently. This is likely because GitVS participants used the main branch visualization while simultaneously listening to sounds, putting emphasis on how the branches, commits, conflicts and developers give rise to meaningful patterns in branch organization. In contrast, GitVS-NS participants looked at either the main visualization or the "details view" about specific commits. This led them to emphasize the details of individual branches, but not on the overall organization and workflow among the branches.

Three GitVS participants mentioned conflicts, as did five GitHub and five GitVS-NS participants. GitHub did not include conflict information, so it is striking that the GitHub participants mentioned conflicts nonetheless. We believe this is because (Q #3) in Phases I & II, refers to con-

flicts, therefore, GitHub participants had already attempted to identify the conflicts and it was on their minds.

In summary, GitHub participants mentioned fewer factors and factors less frequently than participants of GitVS and GITVS-NS. We believe this is because GitVS and GitVS-NS include more information than GitHub. As a result, GitVS and GitVS-NS participants had easier access to more factors and mentioned these when answering Phase-III questions.

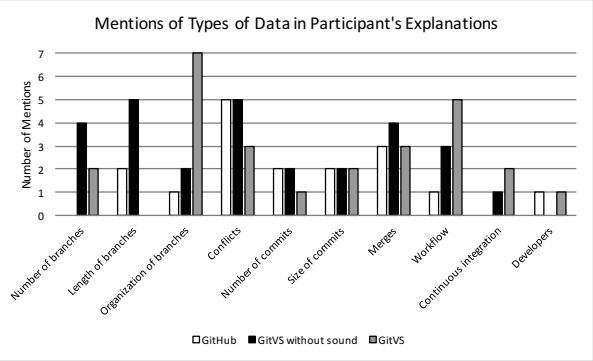

Figure 3: Factors that participants discussed.

6. CONCLUSIONS

We have presented GitVS, a multi-sense view of version control data. In a user study we found that GitVS allows participants to understand version history better than GitHub's network graph, since it presented more information in an easy to access manner. There was no difference in efficiency between GitVS and GitVS-NS.

Sonification was useful when multiple pieces of information was encoded into the view, for example, when understanding the overall workflow and structure across branches. However, when the same information was encoded as a sonification (conflict drums) vs. a visualization (red commit borders), the visualization was more accessible and preferred. This indicates that our central premise holds - sonification is useful when there are multiple dimensions of data to be presented. However, we do need to iterate on how our views and sonification are structured to make it simpler to use.

Since we report on a very small study, and in one case (for GitVS) an outlier has skewed our results we need to perform a larger investigation. We plan to run another, larger study to identify the specific factors where sonification helped and in cases where it failed, to investigate how to encode multi-dimensional data into a single, unified view.

7. ACKNOWLEDGMENTS

We thank Shane Bolan for his help with initial visualizations. This work was supported in part by NSF grants CCF-1253786, HCC-1559657 and CCF-1161767.

8. REFERENCES

[1] J. Atwood. Check in early, check in often. https://blog.codinghorror.com/check-in-early-check-in-often/. Accessed: 2016-03-27.

[2] S. Boccuzzo and H. Gall. Software visualization with audio supported cognitive glyphs. In *ICSM*, pages 366–375, 2008.

[3] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *ESEC/FSE*, pages 168–178. ACM, 2011.

[4] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey. Software history under the lens: a study on why and how developers examine it. In *ICSME*, pages 1–10, 2015.

[5] apache/storm. https://github.com/apache/storm. Accessed: 2016-03-20.

[6] voldemort/voldemort. https://github.com/voldemort/voldemort. Accessed: 2016-03-20.

[7] M. L. Guimarães and A. R. Silva. Improving early detection of software merge conflicts. In *ICSE*, pages 342–352, 2012.

[8] L. Hattori, M. Lanza, and R. Robbes. Refining code ownership with synchronous changes. *Empirical Softw. Engg.*, 17(4-5):467–499, 2012.

[9] T. Hermann, A. Hunt, and J. G. Neuhoff. *The sonification handbook*. Logos Verlag Berlin, 2011.

[10] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey. Dashboards: Enhancing developer situational awareness. In *ICSE(2)*, pages 552–555, 2014.

[11] M. Lanza, L. Hattori, and A. Guzzi. Supporting collaboration awareness with real-time visualization of development activity. In *2010 14th European CSMR*, pages 202–211, 2010.

[12] S. McIntosh, K. Legere, and A. Hassan. Orchestrating change: An artistic representation of software evolution. In *CSMR-WCRE*, pages 348–352, 2014.

[13] K. J. North, S. Bolan, A. Sarma, and M. B. Cohen. Gitsonifier: Using sound to portray developer conflict history. In *ESEC/FSE*, pages 886–889. ACM, 2015.

[14] M. Ogawa and K.-L. Ma. code_swarm: A design study in organic software visualization. *TVCG*, 15(6):1097–1104, 2009.

[15] T. Preston-Werner. Say hello to the network graph visualizer. github.com/blog/39-say-hello-to-the-network-graph-visualizer. Accessed: 2016-03-16.

[16] Processing.org. https://processing.org/. Accessed: 2015-07-17.

[17] R. M. Ripley, A. Sarma, and A. van der Hoek. A visualization for software project awareness and evolution. In *Intl. Works. on Visualizing Soft. for Understanding and Analysis*, pages 137–144, 2007.

[18] A. Sarma, D. F. Redmiles, and A. Van Der Hoek. Palantír: Early detection of development conflicts arising from parallel code changes. *TSE*, 38(4):889–908, 2012.

[19] M. Schonbrun. *Reading Music: A Step-By-Step Introduction To Understanding Music Notation And Theory*. Fall River Press, 2012.

[20] JGit. https://eclipse.org/jgit. Accessed: 2014-12-14.

[21] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov. Quality and productivity outcomes relating to continuous integration in github. In *ESEC/FSE*, pages 805–816. ACM, 2015.

[22] J. Wloka, B. Ryder, F. Tip, and X. Ren. Safe-commit analysis to facilitate team software development. In *ICSE*, pages 507–517, 2009.