Branching and Merging: An Investigation into Current Version Control Practices

Shaun Phillips
University of Calgary
Department of Computer Science
phillist@ucalgary.ca

Jonathan Sillito
University of Calgary
Department of Computer Science
sillito@ucalgary.ca

Rob Walker
University of Calgary
Department of Computer Science
walker@ucalgary.ca

ABSTRACT

The use of version control has become ubiquitous in software development projects. Version control systems facilitate parallel development and maintenance through *branching*, the creation of isolated codelines. *Merging* is a consequence of branching and is the process of integrating codelines. However, there are unanswered questions about the use of version control to support parallel development; in particular, how are branching and merging used in practice? What defines a successful branching and merging strategy? As a first step towards answering these questions, we recruited a diverse sample of 140 version control users to participate in an online survey. In this paper, we present the survey results and 4 key observations about branching and merging practices in software development projects.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: General

General Terms

Theory, Human Factors, Management

Keywords

Version control, source control, revision control, configuration management, branching, parallel development, merging, integration

1. INTRODUCTION

A prominent feature of version control systems is the ability to track the evolution of software along multiple codelines—parallel changes to the same codebase. These codelines are commonly referred to as *branches*. Branching promotes isolated development and maintenance; changes made to one branch will not affect any other branch. This feature allows development teams to partition work and prevent interruptions from external sources. When parallel changes are ready to be integrated, they are done so through *merging*. Merging combines changes from multiple source branches into a single target branch, initiating a conflict resolution process if changes are incompatible.

Many modern version control systems recognize branches as first-class citizens and support a variety of branch specific operations. Distributed version control systems, such as Git or Mercurial, encourage branching as a part of their standard work-flows [7]. However, how branching and merging are actually used in the software industry, and whether these practices are successful, are unanswered questions. In this paper we explore these questions through responses to a broad online survey. From the data, we make 4 key observations, presented as hypotheses, about the use of branching and merging and the effect on parallel development.

CHASE'11, May 21, 2011, Waikiki, Honolulu, HI, USA

2. RELATED WORK

Most version control user manuals describe branching fundamentals and sample usage patterns. A notable example is the Team Foundation Server branching guide [4] which provides practical advice on branching strategies—most of which can be applied to any centralized version control system.

In 1998, Appleton et al. presented a paper that thoroughly describes and classifies a host of branching and merging patterns [1]. The authors also highlight several common branching problems and offer solution advice. A similar article was produced in 2002 by Walrad and Strom [9].

O'Sullivan provides an approachable, albeit brief, description of modern branching practices [7]. Of particular note is O'Sullivan's introduction to branching in distributed systems and how the workflow differs from traditional, centralized approaches.

Williams et al. explore the complexity of merge operations by mining a version control system [10]. The research attempts to understand how developers are using merges, and whether merges are more problematic than other version control operations. Similarly, Mens has summarized and analyzed the different approaches to merging from a technical perspective [5].

A basic web search will reveal many lively discussions on branching and merging "best practices." For example, Linus Torvalds provides merging guidance to Git users in an often-referenced mail thread [8].

However, the above documents and discussions share a common drawback: the justification for using any particular branching or merging strategy is largely anecdotal. Granted, the justification may be grounded in decades of experience,

Table 1: Survey questions. Multiple choice questions are marked with an asterisk.

Demographic
Q1. How many source-level contributors (developers, QA, builders, etc.) are in your product team?
Q2. Which version control system does your product team use? *
Q3. What is the approximate size of your product (in lines of code)? *
Branching
Q4. Does your product team use version control branching? *
Q5. Under what circumstances does your product team create new branches? *
Q6. Please describe your branch creation policy; when, how, and why you create branches.
Q7. What is the topology of your branches? *
Q8. Please describe your branch topology; the general structure of your repository.
Q9. Overall, do you think your product team is using an effective branching strategy? *
Q10. Please provide your thoughts on your branching strategy. What are the advantages/drawbacks?
Merging
Q11. What is your policy on downstream merges (pulling artifacts from parent branches)? *
Q12. Please describe your downstream merge policy.
Q13. What is your policy on upstream merges (pushing artifacts to parent branches)? *
Q14. Please describe your upstream merge policy.
Q15. Typically, how much time is spent on a merge?
Q16. What is the most significant problem you have had with merges? *
Q17. Please explain why the above is your most significant problem.
Q18. Overall, do you think your product team is using an effective merging strategy? *
Q19. Please provide your thoughts on your merging strategy. What are the advantages/drawbacks?
Release
Q20. Does your release plan include time for merging code? *
Q21. Has your branching/merging strategy ever caused a delay in the release schedule? *

but in general the evidence is not comprehensive enough to substantiate claims that certain practices are more conducive to success than others.

Our research distinguishes itself by providing a broad view of current branching and merging practices by surveying an eclectic mix of industry professionals. This process allows us to make key observations about real-world implementations of parallel development and is a step towards identifying the factors that make version control strategies successful.

3. THE SURVEY

From November 10, 2010, to November 26, 2010, we conducted an online survey composed of 21 questions: 3 demographic, 7 branching, 9 merging, and 2 related to release management—all of which are listed in Table 1.

Respondents were recruited from a variety of sources: mailing lists, primarily version control system-specific, though one was targeted towards organizations that employ agile practices; online forums used for system-specific version control discussions; and through industry contacts. Because of the anonymity provided by many of these sources, it is difficult to estimate the total number of individuals who viewed the recruitment advertisement. However, we could track the actions of respondents after they entered the survey, and this participation is broken-down as follows:

- Respondents that entered the survey: 312
- # Complete: 140 (45%)
- # Partial or incomplete: 70 (22%)
- # Abandoned (no data entered): 102 (33%)

There was no feedback mechanism to capture why some surveys were abandoned or left incomplete, but nevertheless, only responses from complete surveys were considered for this study. Here, a "complete" survey was one in which all required questions had responses (Q1-Q4, Q9, Q15, Q16, Q18). The remaining questions were optional, as they needed insight into organizational practices that may not be possessed by all respondents. But in most cases, respondents proved to have this institutional knowledge. The aggregated respondent demographic information (Q1-Q3) from the complete surveys is depicted in Table 2.

When constructing the survey, we employed both multiple choice and open-ended questions. Using multiple choice questions, for example, "What is the most significant problem you have had with merges?" (Q16), simplifies analysis and reporting. However, because this is an exploratory study, sole reliance on multiple choice questions would limit the insight we could gain from the respondents. Thus, the survey also included open-ended questions; for example, the follow-up to Q16 was: "Please explain why the above was your most significant problem" (Q17). The intent of these questions was to enable respondents to put their multiple choice selections into context, identify unanticipated factors, and to help us better understand patterns or anomalies in the data.

3.1 Limitations

It should be noted that, as with all online surveys of this nature, there are limitations to the study. In particular, the respondents, while demographically diverse, were not part of a random sample—participation was self-selected and vol-

Table 2: Demographic Information of Survey Respondents

Version Control System Used		# of Source Level Contributors		Product Sizes	
System	# Respondents	# Contributors	# Respondents	Product Size (LOC)	# Respondents
AccuRev	2 (1%)	1-10	67 (48%)	< 100,000	37 (26%)
Bazaar	11 (8%)	11-20	32 (23%)	500,000	32 (23%)
ClearCase	2 (1%)	21-40	13 (9%)	1,000,000	21 (15%)
CVS	11 (8%)	41-60	7 (5%)	10,000,000	18 (13%)
CVSNT	2 (1%)	61-80	2 (1%)	25,000,000	6 (4%)
darcs	5 (4%)	81-100	4 (3%)	50,000,000	7 (5%)
Git	30 (21%)	101-200	5 (4%)	100,000,000+	8 (6%)
Git-SVN	2 (1%)	201-500	4 (3%)	Unknown	11 (8%)
Mercurial	24 (17%)	501-1000	3 (2%)		
Subversion	31 (22%)	1000+	3 (2%)		
Synergy	1 (1%)				
TFS	19 (14%)				

untary. Because of this inherent bias, care should be taken when generalizing the results.

Additionally, there are many version control systems in use today, and their implementations of branching and merging vary. As an example, a "cloned" repository in Mercurial may be used as a branch, even though there is separate branching functionality available. Likewise, version control vocabulary can differ between systems and organizations: even the term "version control" is not standardized, with "source control", "revision control", and "source configuration management" being accepted alternatives (although they sometimes have subtly different meanings). So, in order to make the survey accessible to a broad audience, some questions were crafted generically. If these questions were oversimplified, there is a risk that the collected data may not include specific, but relevant information. However, this risk is mitigated by our use of open-ended questions, which provide an outlet for this data.

Finally, the options for the multiple choice questions were created by the authors of the study, albeit motivated by personal experience and previous literature, and refined by survey piloting feedback. Nonetheless, there is a risk that the available options did not cover the spectrum of relevant choices. This concern, however, is again mitigated by the open-ended questions, where respondents can provide feedback not present in the multiple choice options.

4. INITIAL FINDINGS

Given the preliminary nature of this study, the primary focus is on the responses to the multiple choice branching and merging questions, which are supported by notable respondent comments. The full analysis of the open-ended responses will be presented in a future study. Likewise, the data related to the software release process (Q20 and Q21) will be addressed in subsequent work.

4.1 Branching

Here, we present the results of the multiple choice branching survey questions (Q4, Q5, and Q7). Q9, regarding overall branching satisfaction, will be discussed in section 5.

4.1.1 Drawbacks to Branching

Despite facilitating parallel development, there is a significant drawback to branching. Namely, as changes are introduced, branches will diverge and merges become increasingly complex. This phenomena is referred to as "integration hell" [3]—the arduous process of piecing together disparate software components. As articulated by a survey respondent:

> "We are a team of four senior developers (by which I mean we're all over 40 with 20+ years each of development experience) and not one of us has had a positive experience in the past with branching the mainline...The branch is easy - it's the merge at the end that's painful".

This is a common sentiment from respondents who view branching unfavourably. Poor branching support is also seen as a barrier to use; in particular, branching in some systems is considered slow or complicated. In total, 12 respondents (9%) indicated they do not use any form of branching in their projects. These 12 respondents were thus excluded from answering the remaining questions, leaving 128 complete responses for analysis.

4.1.2 Branch Types

To identify how branches are being used in development projects, we asked respondents to indicate the specific purposes for which they create branches. In this context: *Release* branches are used to maintain specific versions of a product; *Experiment/prototype* branches isolate disruptive or unstable code that may not go into production; *Feature* branches are primarily used to partition product development into cohesive components; *Bug fix* and *merge* branches, as their names imply, are created for specific tasks in the development cycle; *Contributor* branches are typically used to isolate individual work before it is shared with the product team; *Platform/SKU* branches are used to separate different product configurations or contexts (e.g., x86 vs. x64); finally, *Tool/utility* branches prevent changes to critical development, test, and build processes from disrupting workflow. Table 3 shows the proportion of respondents using each type of branch.

Table 3: Branch Types Created (Q5)

Branch Type	Number of Respondents
Release	93 (73%)
Experiment/Prototype	90 (70%)
Feature	82 (64%)
Bug Fix	53 (41%)
Contributor (Personal)	46 (36%)
Merge	37 (29%)
Tool/Utility	18 (14%)
Platform/SKU	16 (13%)

The data shows that branch-per-release and branch-per-feature models are the most popular amongst respondents. Similarly, branching for higher-risk activities, such as experiments or prototypes, are commonly used models.

An interesting facet of the data is the use of small-scope, short-lived branches. Bug fix and merge branches are typically created for single, targeted purposes, and the use of these branches is strongly correlated with the type of version control system being used. That is, 85% of respondents who employ both models are using distributed systems. Distributed version control is often referred to as "branchy" [7] as it is felt the systems have minimal overhead associated with branch creation, thus promoting the use of short-lived branches. Our findings, at the very least, do not contradict these claims.

4.1.3 Branch Topology

Branch topology describes the layout and structure of the branches in a version control system. If we conceptualize the system as a tree, the root node would be the mainline branch. In a *flat* topology, all child nodes (branches) have a depth of 1. That is, all branches are directly connected to the mainline branch. In a *staged* topology, child nodes can be at an arbitrary depth—branches may have parents other than the mainline. For example, a "Release 1.1" branch may have "Release 1.0" as a parent branch, rather than the mainline.

Staged topologies offer more opportunities to perform integration testing and to transfer responsibility before changes reach the mainline branch. However, greater effort is required to propagate changes around the tree. Flat topologies are seen as more agile, because changes are only one merge away from the mainline; however, there may be a greater risk of destabilizing the project. Table 4 shows the branch topologies used by the survey respondents.

Table 4: Branch Topologies (Q7)

Topology	# of Respondents WRT Project Size		
	All Sizes	1M+ LOC	10M+ LOC
Flat	74 (57%)	26 (46%)	15 (42%)
Staged	44 (34%)	25 (45%)	19 (53%)
Unknown	11 (9%)	5 (9%)	2 (5%)

If we filter the data by project size, the use of staged topologies is increasingly more common. Specifically, if we consider projects with at least 1,000,000 lines of code, staged use is 45%. And in projects with at least 10,000,000 lines of code, usage climbs to 53%. However, several respondents from these large projects cited problems with their topologies:

> "Branches for releases are critical and an excellent idea. Branches for teams and features can help isolate problems, but the resulting integration costs and stabilization times are too high. I think using effective [test] suites and an automated check-in system with a single branch would be a much more efficient way for us to work".

> "We have a large project with many dependencies between teams. One challenge of having team branches and nested sub teams and feature branches is that the propagation delay can take a long time for dependent teams to receive features".

Thus, the survey data shows that large projects are more likely to stage their branches, but face difficult trade-offs in propagation errors and delays.

4.2 Merging

Here, we present the results of the multiple choice merging survey questions (Q11, Q13, and Q16). Q19, regarding overall merging satisfaction, will be discussed in section 5.

For this study, merges are distinguished by their directionality, as the corresponding policies can differ substantially. *Downstream* merges pull changes into lower branch levels, typically from parent to child, e.g., from the mainline into a feature branch. Conversely, *upstream* merges push changes into higher branch levels, typically from child to parent, e.g., from a feature branch into the mainline.

4.2.1 Downstream Merges

Downstream merges may be required due to feature dependencies, but keeping a branch in sync with its parent also ensures that testing remains relevant and allows integration defects to be caught sooner. Moreover, performing downstream merges regularly reduces the scope and impact of conflicts on the eventual upstream merge. As a consequence, however, a branch can frequently destabilize if significant changes are introduced.

> "...the problem is that the merges disrupt productivity for developers. A way needs to be found to update developers to the latest code painlessly and quickly".

Several downstream merge policies can be adopted: *Ad-hoc* approaches merge code when needed or convienient, but not on a set schedule; *Event-driven* policies merge changes into branches on significant product milestones, such as a beta or test pass; and *Periodic* approaches merge changes according to a regular schedule to ensure branches do not excessively diverge. Table 5 shows the downstream merge practices of the survey respondents. Multiple policies can be in effect, so respondents could select more than one option.

Policy usage is relatively consistent across product sizes, and clearly the most popular approach is ad-hoc. Periodic policies, while the least common selection, are slightly more common in larger products.

Table 5: Downstream Merge Policies (Q11)

Policy	# of Respondents WRT Product Size		
	All Sizes	1M+ LOC	10M+ LOC
Ad-hoc	97 (76%)	41 (73%)	27 (75%)
Event	44 (34%)	25 (45%)	17 (47%)
Periodic	22 (17%)	16 (29%)	11 (31%)

4.2.2 Upstream Merges

Merging upstream is a higher risk activity than downstream; if the changes are destabilizing, then the impact can affect the the entire product team. On the other hand, many of the benefits of regular downstream merges can not be achieved without the regular sharing of changes.

Similar to downstream policies, *Event-driven* approaches merge upstream on significant branch events, such as the end of a sprint or milestone. Likewise, *Periodic* policies share changes on a regular cadence. *On-Completion* approaches will only merge upstream when the work in the branch is fully completed. Table 6 depicts the upstream merge practices of the survey respondents. Again, because multiple policies can be in effect, respondents could select all applicable policies.

Table 6: Upstream Merge Policies (Q13)

Policy	# Respondents WRT Product Size		
	All Sizes	1M+ LOC	10M+ LOC
On-Completion	85 (66%)	36 (64%)	22 (61%)
Event	57 (45%)	23 (41%)	15 (42%)
Periodic	13 (10%)	6 (11%)	5 (14%)

Respondent selections were again roughly consistent across product sizes. Policies that merge less-frequently, specifically, on-completion and event-driven, were the favoured approaches–perhaps highlighting the caution taken to not destabilize the product. These choices were reflected in respondent comments:

> "We merge upstream when and only when the upstream maintainer is happy with the work. Upstream has multiple downstreams so quality control needs to be very high".

> "Only when things are reasonabl[y] stable are they merged into parent branches. But that depends on the parent branch. Development code is done very frequently. Release/production trunk are done rarely."

The latter quotation raises an important point: merge policies often depend on the branch types involved in the merges. For example, a short-lived bug fix branch will merge on-completion and have minimal scope. However, if a feature branch only merges on-completion, the scope can be much larger and the time frame much longer. These differences were not captured in the quantitative question.

4.2.3 Merging Problems

Integration hell is an extreme example of branching gone awry. Nevertheless, even successful branching and merging strategies will encounter problems. In the survey, we wanted to capture the most significant problem encountered by the respondents as a result of their version control strategies.

Several common merge problems were presented as options in the survey (Q16); namely, *Merge Conflicts*, which occur when parallel changes to the same areas of code are incompatible; *Test Regressions* refer to failures in a test suite during or after a merge; *Cross-Cutting Regressions* are problems that affect multiple parts of the product and are difficult to capture in a test suite, such as performance or security degradations; *Loss of Productivity* issues arise when merges disrupt normal developer workflow, especially if a branch destabilizes after new changes are introduced; finally, *Compilation Errors* result from improper merge conflict resolutions that produce syntactically incorrect code, or from errors introduced into the build scripts themselves. Additionally, respondents were given the opportunity to select an "other" option and asked to qualify their most significant perceived merge problem.

Table 7 aggregates the responses both from all respondents, and from those from large, active projects (at least 40 contributors and 10,000,000 lines of code).

Table 7: Most Significant Merge Problem (Q16)

Merge Problem	# of Respondents	
	All Projects	Large, Active
Merge Conflicts	69 (54%)	6 (32%)
Other	20 (16%)	2 (11%)
Test Regressions	17 (13%)	2 (21%)
Cross-Cutting Regressions	12 (9%)	4 (21%)
Loss of Productivity	7 (5%)	2 (10%)
Compilation Errors	3 (2%)	1 (5%)

The data shows that merge conflicts are seen as the most significant problem associated with merging. However, our use of the term "significant" is somewhat ambiguous and can interpreted as the most frequently occurring problem, or perhaps an infrequent but hugely disrupting problem. The accompanying comments in the open-ended follow-up question (Q17) help us put some of these selections into context:

> "Merge conflicts are tedious to fix manually and often the mechanic[al] approach makes one miss the one crucial change that should have been kept."

> "Usually in non source code file[s] the merge conflicts are a pain."

> "Code complexity and the project size can make the resolution process quite complex, especially when the platforms changes [sic]."

Respondents who selected "other" typically specified the loss of historical revision data after merges, or they used this response to indicate they had no perceived merge problems.

An interesting pattern arises when we consider the selections of respondents from large, active projects. First, it should be noted that both test and cross-cutting regressions are problems associated with product quality. In this subset of respondents, we see that issues with product quality are the most reported merge problem—a combined 42% (vs. 32% for merge conflicts). This finding may imply that merge-specific product quality regressions have an increased significance as an organization grows in size.

5. KEY OBSERVATIONS

In this paper, the relationships between the quantifiable metrics from the survey are the focus of the analysis. Interesting patterns have emerged, and we feel that our observations support the formulation of 4 hypotheses. In this section, we describe each hypothesis, as well as suggest approaches to test them in future work.

Continuous integrations are typically not done in practice. Continuous integrations are often touted as the solution to integration hell, particularly from those in the agile community. Duvall et. al. have written a book on the subject [3], and there has been recent case study showing positive results when a company implemented a continuous integration process [6]. Intuitively, this would be an effective model: we avoid branch divergence and its associated drawbacks. However, results from our survey indicate that continuous integrations are not being used consistently in practice.

Only 7/128 (5%) of respondents reported using both periodic upstream and downstream merges, a notable minority. Certainly, it may be the case that some respondents use only short-lived branches and merge on-completion, which is a form of continuous integration. Likewise, because no question specifically stated "continuous integration", we may have missed similar scenarios.

Nevertheless, when considering the data comprehensively, there is little evidence that many respondents are merging in a continuous manner. As we have shown, the majority of created branches are long-lived, 35–50% of topologies are staged, and merge policies tend to be ad-hoc, completion, or event-driven.

Barriers to using continuous integrations can possibly be the higher risks of destabilization, and the loss of productivity due to branching/merging overhead; both problems were cited in respondent comments. For future work, we can test the hypothesis with the following questions:

- Do you perform continuous integrations from development branches?
- What prevents you from merging more frequently to and from development branches?

Successful branching and merging strategies focus on reducing the frequency and complexity of merge conflicts, as well as preventing product quality regressions during merge operations. Respondents from small projects, either in product size or the number of contributers, typically reported fewer overall problems with their branching and merging strategies. In these situations, change and branch divergence will occur at a slower rate, and version control practices will have less overall impact.

Nevertheless, the majority of respondents, even those from small projects, indicated that merge conflicts were their most significant, or at least the most common, problem. So, successful strategies will then have some method of reducing the impact of merge conflicts. We can test this part of the hypothesis with two follow-up questions:

- What is your conflict resolution process?
- Which tool(s) do you use to resolve conflicts?

However, the survey data has also indicated that as a project grows in size and activity, concerns about product quality resulting from merges become more significant. These projects typically produce changes at a rate that cause branches to quickly diverge. Subsequently, merges may then be complex and error prone—producing unintended defects.

Thus, successful branching and merging strategies must also take steps to prevent, or efficiently detect, defects that result from the merge process. This facet of the hypothesis can be tested through follow-up questions that explore testing methodologies. Specifically:

- At which point(s) in the merge process do you test?
- What types of tests do you run?
- What is your testing process during merges?

Branching satisfaction is influenced by the types of branches created, with the use of feature, release, and experimental branches having the most impact. The choice of version control system has little correlation with branching satisfaction. Respondents were asked whether they were generally satisfied with their branching practices. In Table 8, we applied the Naive Bayes Algorithm, a common data-mining technique with tabular data, to the results to identify the key influences on the responses to this question. The "relative impact" for a particular response is a predictive score about its influence on the target outcome.

Table 8: Influences on Branching Satisfaction

Survey Response	Relative Impact
Uses Experiment Branches	79
Uses Feature Branches	44
Uses Release Branches	39
Uses Staged Topology	10
Uses Distributed System	9

We note that the usage of three branch types have the greatest impact, most notably, high-risk experiment and prototype branches. One method to test this hypothesis would be to expand the data-set by recruiting more survey respondents and verifying that the claim continues to hold. That is, the Naive Bayes Algorithm performs better with large amounts of data, and we can observe how the relative impact factors change (if at all) with a larger response pool. Alternatively, the hypothesis can be tested by examining the statistical significance of these correlations from the survey data.

In general, there appears to be no correlation between the choice of version control system and branching satisfaction. Individual comments often refer to specific problems in particular systems, but when considered comprehensively, no patterns are apparent. Thus, the findings indicate that effective branching strategies can be achieved regardless of the system used. This assertion can be tested with the following question:

- Are you satisfied with the branching facilities provided by your version control system? If no, why not?

Merging satisfaction is influenced by the frequency of upstream merges, with the use of upstream periodic and event-driven merges having the most impact. The choice of version control system is highly correlated with merging satisfaction. As with branching satisfaction, in Table 9 we applied the Naive Bayes Algorithm to find influences on merge strategy satisfaction:

Table 9: Influences on Merging Satisfaction

Survey Response	Relative Impact
Uses Distributed System	100
Uses Upstream Event Merges	85
Uses Upstream Periodic Merges	17

Here, the use of distributed systems and frequent upstream merges have the most impact over merging satisfaction. Regular upstream merges reduce the risks associated with branching and the complexities of the individual merges. While recent research has indicated that merging is "simple" in distributed systems [2], it is not clear exactly why distributed systems lead to greater merge satisfaction over their centralized counterparts. We can further explore and test this hypothesis both quantitatively and qualitatively. First, we can recruit more survey respondents to see if the pattern is still present. Second, we can explore the significance of the correlations through statistical tests. Finally, we can present the following question to appropriate audiences:

- How has using a distributed system affected your merge process, both before and after the transition from a centralized system?

6. CONCLUSIONS

Parallel development can be achieved through the branching features of version control systems. However, how branching and merging are actually employed in the software industry is largely unstudied. To address this deficit, we have formulated several branching and merging hypotheses from data generated by a comprehensive survey. The survey recruitment was successful in both the number of responses received and the diversity of respondents, but as with all results from online surveys, care should be taken when generalizing the results. Yet, because this was an exploratory study, generalization is not the goal. Instead, we want to construct theories that are grounded in real data. In this regard, the success of the recruitment campaign adds credibility to our hypotheses.

We have outlined how to further test each hypothesis in future work, which may be another survey or a set of interviews. The goal of continued research is to establish best practice guidelines for organizations that choose to branch their development projects. There remains a wealth of information to be analyzed from the current data-set, however, from the open-ended responses to correlations with release management. Nevertheless, the contributions of this study, namely, the descriptions and the preliminary analysis of the survey responses, can assist version control researchers, developers, and tool makers in their future work.

7. REFERENCES

[1] B. Appleton, S. Berczuk, R. Cabrera, and R. Orenstein. Streamed lines: Branching patterns for parallel software development. In *5th Annual Conference on Pattern Languages of Program Design*, 1998.

[2] B. de Alwis and J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, CHASE '09, pages 36–39, Washington, DC, USA, 2009. IEEE Computer Society.

[3] P. Duvall, S. Matyas, and A. Glover. *Continuous integration: improving software quality and reducing risk*. Addison-Wesley Professional, first edition, 2007.

[4] B. Javidi, J. Pickell, B. Heys, T. Erwee, and W.-P. Schaub. *Microsoft Visual Studio Team Foundation Server Branching Guidance 2010*. Microsoft Corporation, 2010 edition, 2009.

[5] T. Mens. A state-of-the-art survey on software merging. *Software Engineering, IEEE Transactions on*, 28(5):449 –462, May 2002.

[6] A. Miller. A hundred days of continuous integration. In *Agile, 2008. AGILE '08. Conference*, pages 289 –293, 2008.

[7] B. O'Sullivan. Making sense of revision-control systems. *Queue*, 7:30:30–30:40, August 2009.

[8] L. Torvalds. [git pull] drm-next, March 2009. http://www.mail-archive.com/dri-devel@lists.sourceforge.net/msg39091.html.

[9] C. Walrad and D. Strom. The importance of branching models in scm. *Computer*, 35:31–38, September 2002.

[10] C. C. Williams and J. W. Spacco. Branching and merging in the repository. In *Proceedings of the 2008 international working conference on Mining software repositories*, MSR '08, pages 19–22, New York, NY, USA, 2008. ACM.