A Theory of Branches as Goals and Virtual Teams

Christian Bird
Microsoft Research
Redmond, WA
cbird@microsoft.com

Thomas Zimmermann
Microsoft Research
Redmond, WA
tzimmer@microsoft.com

Alex Teterev
Microsoft
Redmond, WA
alextet@microsoft.com

ABSTRACT
A common method of managing the complexity of both technical and organizational relationships in a large software project is to use branches within the source code management system to partition the work into teams and tasks. We claim that the files modified on a branch are changed together in a cohesive way to accomplish some task such as adding a feature, fixing a related set of bugs, or implementing a subsystem, which we collectively refer to as the goal of the branch. Further, the developers that work on a branch represent a virtual team. In this paper, we develop a theory of the relationship between goals and virtual teams on different branches. Due to expertise, ownership, and awareness concerns, we expect that if two branches have similar goals, they will also have similar virtual teams or be at risk for communication and coordination breakdowns with the accompanying negative effects. In contrast, we do not expect the converse to always be true. In the first step towards an actionable result, we have evaluated this theory empirically on two releases of the Windows operating system and found support in both.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Version Control*; D.2.9 [**Software Engineering**]: Management – *Software Configuration Management.*

General Terms
Management, Measurement, Human Factors

Keywords
Branching, Teams, Coordination

1. INTRODUCTION
Software development in any large project is a collaborative and team-based enterprise. Disorganization in such contexts is known to lead to delays [1] and faults [2]. An ideal solution to the problem of organizing a large software effort is to decompose the system into highly cohesive and loosely coupled modules [3] and create teams around these modules, leveraging Conway's Law [4]. We hasten to note that Parnas' definition of a module may not characterize the conventional definition. From his paper, "In this context 'module' is considered to be a responsibility assignment rather than a sub-program." In practice, we have observed that development teams face barriers when attempting to organize in this way. Some "modules" are cross-cutting and defy loose coupling with the rest of the system. Other modules may be highly dependent on other components within the system. In both cases, teams may need to be insulated from the changes of others that may destabilize parts of the system that it is working or dependent on.

Once the decomposition of the system into modules has been decided upon and resources (in the form of developers) have been assigned tasks, how do they perform their work in such a way that they can share with each other but remain isolated during times of rapid and volatile development? A common solution to this problem is through the use of branches within the software configuration management (SCM) system. Using branches allows teams to defer integration and can insulate them from the changes of others that may hinder their own progress.

Branching within an SCM allows multiple teams to create their own workspaces (usually called a branch) from a particular state of the source code. Each team commits to their own branch as they normally would in their SCM and at some point in the future, once their tasks have been completed, the changes in their branch are integrated (also known as merged) into the trunk or a release branch. The effort involved in such an integration is usually dependent on how much work went on in the branch and also in the original branch in the intervening time. By providing isolation, branches allow teams to focus on their own tasks without prematurely worrying about or being affected by the changes occurring in the rest of the system. Teams are free to explore, develop, test, and stabilize the code base without interfering with others.

Recently, many open source software (OSS) projects have begun using branches more heavily as a result of moving to more advanced SCMs such as Git and mercurial [5]. Within Microsoft, branching is used heavily to insulate teams from each other's possibly unstable changes. In both contexts, we have observed anecdotally that a branch embodies a goal (we use a purposely vague term) which may represent one or more related features, user scenarios, or subsystems and a virtual team that represents those contributors working on this goal.

We claim that the SCM is the most important collaborative tool used during software development because it is the mechanism by which developers actually share the technical artifact. As such, their collaborative behavior as evidenced by SCM records is a valuable resource and tracking this behavior with an expectation of what is generally "good" and "bad" may enable project managers to help their teams avoid problems. We follow in the steps of others by examining developers' behavior through the lens of historical repository data [6].

Prior research suggests that awareness is important as developers work in different branches (also referred to in other work as

CHASE'11, May 21, 2011, Waikiki, Honolulu, HI, USA
Copyright 2011 ACM 978-1-4503-0576-1/11/05… $10.00.

workspaces) [7]. Based on findings that awareness is important in large projects [8] and that lack of such awareness in software development can have detrimental effects [9], we assert that in certain scenarios, there may be adverse effects due to the insulation that branches afford software teams. In this paper, we present a theory of branches as goals and virtual teams and explore this theory by evaluating it on two releases of the Microsoft Windows operating system.

2. THEORY

Branches are most often created to accomplish some goal, which may be implementing a feature, performing a maintenance exercise, continued maintenance on a subsystem, or fixing a number of related bugs. Each goal has a profile that is defined by the files that need to be changed to accomplish that goal. In addition, the goal has a virtual team embodied by the developers who make the changes required to realize the goal. However, files may (and often do) take part in multiple goals and people may also help in accomplishing multiple goals as well. We use branches to identify the file and author profile for each goal. Prior research has shown that when different sets of developers or teams are working on the same parts of the system without adequate communication, tasks take longer to complete [1] and defect rates increase [2]. We therefore claim that if two branches have a high similarity in terms of the files modified to accomplish that goal, then there must be a corresponding high similarity in terms of the developers working in those branches. If not, there is risk of low communication between disparate teams working in the same systems for possibly competing or conflicting goals. Consequences of such scenarios could include syntactic or semantic conflicts, incompatible design changes, duplication of work, or longer stabilization phases.

$$\textit{High Goal Similarity} \Rightarrow \textit{High Virtual Team Similarity}$$

However, we do **not** expect that the converse is true. The same or similar teams may work on unrelated goals without a negative impact on productivity or quality. People may have expertise in multiple areas without undue negative effects; files generally fare better when their functionality is focused and internally cohesive.

$$\textit{High Virtual Team Similarity} \nRightarrow \textit{High Goal Similarity}$$

3. EMPIRICAL EXPLORATION

We evaluate our proposed theory in this paper by examining these relationships in branches within two releases of Windows. These releases contained developers and binary components both numbering in the thousands. Ideally, our research consists of three steps. First, we need to verify that development actually follows this pattern. That is, is it empirically the case that branches with similar goals are worked on by similar teams? Second, we must demonstrate that adherence (or the lack of adherence) to this theory is tied important outcomes. In software engineering, two of the most important outcomes are quality and productivity. Quality is usually measured in terms of defects, bugs, post-release failures, and regressions. Productivity is often measured in tasks completed per unit time or the length of time needed for a system to stabilize. Are these negatively impacted when branches have similar goals but disparate teams? Third, provided that prior steps yield positive results, we plan to develop a tool that can alert stakeholders to violations of our branch similarity theory and evaluate if such a tool is able to improve outcomes by improving adherence to our theory.

This paper is exploratory in nature and addresses the first step of the aforementioned research plan by evaluating how often practice matches our theory of branches as goals and virtual teams.

3.1 Methodology

To evaluate our theory of Branches empirically, we gathered historical information from two releases of the Windows Operating System, Windows Vista and Windows 7. The Windows codebase is developed by engineers working on a number of different branches. These branches form a general tree structure (there are occurrences of changes moving from one branch to an unrelated branch, but these are uncommon) which converges at a trunk from which Windows is released. Although this forms a hierarchy a number of levels deep, for this preliminary examination, we do not differentiate by branch depth. For our purposes, we gathered all changes that were made on all branches during the development cycle for both releases of Windows (analyzed individually) and partitioned them by the branches where the changes originated. We then compared all pairs of branches by looking at the similarity between *who* worked on the two branches and the similarity between *what* was changed on the two branches.

Upon manual inspection we discovered that most branches modify many files, but a few files in each branch are modified an order of magnitude more than the others. Dichotomizing the data by using only sets of changed files (required when using a set similarity measure such as Jaccard similarity) and ignoring the number of changes per file squanders this vital information. We therefore leverage this information and use *cosine similarity* [10] which measures the cosine of the angle between two vectors in an n-dimensional space and ranges from 0 to 1.0, with 1.0 indicating exact similarity and 0 indicating there is no intersection of non-zero dimensions in A and B.

$$\textit{cosine similarity}(A,B) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Formally, we describe our approach as follows. Let D be the set of developers, F be the set of source files modified, and B be the set of development branches all during a release of Windows. Note that F may include files that contribute to tests, build infrastructure, or tools that are considered part of Windows development but that do not actually compile into any portion of Windows that ships. We characterize each branch, $b \in B$, by two vectors. One vector, b_D, represents the contributors to b, with dimension $\|D\|$. Each dimension of b_D represents one developer and the value for branch b is the number of contributions (i.e., individual changes, also known as commits or checkins) made by that developer on b. This characterizes the set of developers working on a branch and we term this set a *virtual team* because it may span multiple organizational teams. Similarly, the second vector, b_F, with dimension $\|F\|$, represents the files changed in b. Each dimension in b_F corresponds to a source file and its value is the number of changes to that file on b. We call this vector the *goal profile* for b. We are interested in identifying and characterizing those pairs of branches that have similar goals. Once we obtain file and developer vectors for two branches, we compute their similarity scores using cosine similarity.

We calculated the team and goal similarity for all possible pairs of branches and examined the relationship between these similarities in two ways. First, we generated a visual depiction of these similarity relationships to determine if any visible trends were apparent in the data. Figure 1 shows a heatmap of the relationship

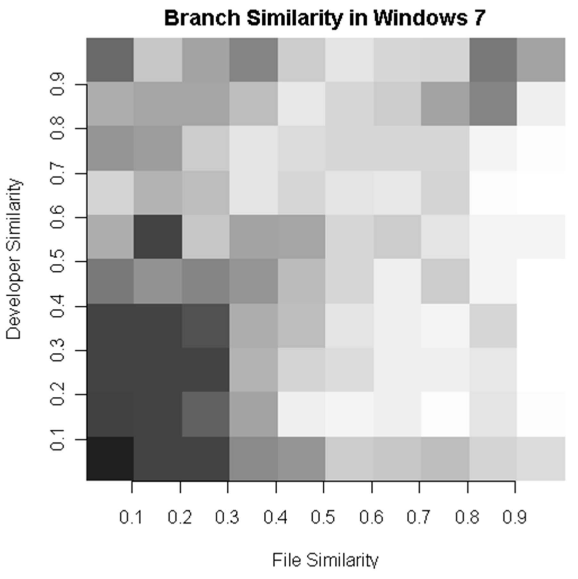

Figure 1. Branch Similarity in Windows 7. Darker cells have more branch pairs with the indicated file and developer similarity. Lighter cells in the bottom right and darker cells on the top and left support out hypothesis.

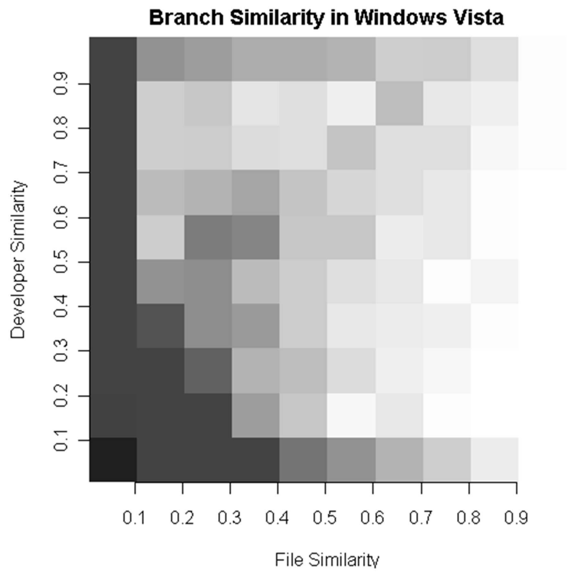

Figure 2. Branch Similarity in Windows Vista. Most darker cells are in areas where developer similarity is higher than file similarity, indicating that our hypothesis is supported in Windows Vista.

between these similarities. Each pair of branches has a team similarity and a goal similarity, and can thus be characterized by one point on a plane. Each cell in the heatmap is shaded according to the number of points that fall within the confines of that cell (essentially, colored according to density). Darker values indicate that many branch-pair similarities fell in that cell.

While a visual depiction is useful for observing goal and team similarity, we also perform a rigorous statistical analysis. It is unclear how to test that an implication is true to a statistically significant degree. A correlation of goal similarity with team similarity can give some sense of the relationship, but may not completely capture it because we do not expect that high team similarity will always lead to high goal similarity; the same team may work on disparate goals. A correlation will only indicate if high values of one are always associated with high values of the other and low values of one are always associated with low values of the other. Concretely, a correlation can validate:

$$\textit{High Team Similarity} \Leftrightarrow \textit{High Goal Similarity}$$

This is in fact not what we are asking. Nonetheless, we use correlation as it provides a lower bound on the strength of the implication relationship. In our case, we use a Spearman rank correlation [11] because both similarity distributions are highly right skewed (most mass is on the left); the majority of branch pairs have very low similarity in both virtual teams and goals.

To augment this analysis, we also define an inequality relationship between team similarity and goal similarity. If high goal similarity generally leads to high team similarity then we would expect the following inequality to hold.

$$\textit{Virtual Team Similarity} > \textit{Goal Similarity}$$

We then examine what proportion of branch pairs maintains this inequality relationship. We don't expect that all branch pairs will exhibit this property, but we do expect that the majority will. In order to test this statistically, we use a binomial test [12]. This test treats each pair of branches as an individual Bernoulli trial that is true if the inequality holds and false otherwise. It indicates if the inequality is true the majority of the time to a statistically significant degree.

To control for false discovery, a phenomenon where multiple statistical tests on the same data may result in spurious statistically significant results, we use Benjamini-Hochberg correction of p-values [13].

3.2 Findings & Discussion

Figure 1 shows the branch similarity heatmap for the branches used in development for Windows 7. This visualization provides evidence supporting our theory. First, note that the cell with the most similarities has developer similarity and file similarity values both between 0 and 0.1. This is because on average, most branches are not similar to each other in either virtual team or goal space. Of interest are the two portions of the heatmap divide by the diagonal $x = y$ line. The cells below the $x = y$ line (bottom and right) are far less dense than the cells above the $x = y$ line (top and left). This indicates that most of the time, virtual team similarity is at least as high as goal similarity for a pair of branches. If two branches share a similar goal, they tend to have similar virtual teams of developers working on them. This relationship is not, however, converse. There are cells with high density near the top left, which represents many pairs of branches with high virtual team similarity, but not high goal similarity. Just because the virtual teams on two branches are similar does not mean that their goals will (or should) be similar.

While this visual examination appears to support our hypothesis. We also evaluated our hypothesis using statistical inference. For Windows 7, the Spearman rank correlation of team similarity with goal similarity was 0.39 and a test of significance yielded $p \ll 0.01$ indicating a statistically significant, but moderate effect. One reason why this correlation may not completely capture the entire story is that there are many pairs of branches that have high team similarity but low goal similarity. This lowers the correlation value, but is in fact in line with our hypothesis that high goal similarity will lead to high team similarity.

We also examined cases of pairs of branches that had either (or both) high team or high goal similarity. Within Windows 7, we found of the branch pairs that are similar in terms of virtual teams or goals, 68% have higher team similarity than goal similarity. A binomial test showed that this is a statistically significant result as well, with $p \ll 0.01$.

Figure 2 shows the heatmap for Windows Vista. Again, observe that the density of branch pairs above the $x = y$ line is higher than below, indicating that it is rare for branches to be similar in terms of goals, but not in terms of the virtual teams accomplishing them. The Spearman correlation of goal and virtual team similarity in was 0.47 with $p \ll 0.01$ indicating a slightly higher, but still moderate and statistically significant correlation between similarities. In our analysis of how often team similarity is higher than goal similarity in Vista, team similarity was higher 75% of the time, again with a binomial test significance of $p \ll 0.01$.

Our empirical analysis indicates that our theory of branch similarity in terms of goals and teams is supported by the development activity in the last two releases of Windows.

4. RELATED WORK

Space limits our discussion of the wealth of prior work related to awareness, collaboration of software teams, and use of branching. Sarma *et al.* developed [14] and evaluated [15] *Palantir*, a tool for workspace awareness. While we are not aware that Palantir has been evaluated in the context of branches in source code management systems, fundamentally there is no reason that it would not work in such a setting. Palantir addresses the same high level concern as our own, awareness between contributors to software projects. Guimaraes *et al.* implemented a novel continuous integration system, *WeCode*, to be employed by teams using branches to collaboratively deal with integration issues early and improve checkin quality [7].

Tools such as Palantir and WeCode are aimed at mitigating existing issues and identify very concrete problems *after* or *as* they occur. In contrast we are interested in identifying disparate teams working on the same parts of the system that have the potential to make changes which may conflict at the syntactic or, much worse, the semantic or design level. Our target audience differs in that we aim to provide awareness to project management rather than developers, so that they may avoid these scenarios or coordinate these efforts without adverse effects. As soon as a virtual team knows which files need work to realize their goals (as early as design time) management may use such information to identify which virtual teams may require additional coordination.

5. LOOKING FORWARD

These results represent the first step in our exploration of the relationship between teams and goals on branches. While encouraging for this line of research, more must be done for it to be useful and actionable. In this paper, we have proposed a theory and have found that it generally holds across two releases of Windows. We have not, however, shown that the results of software development are any different when this hypothesis holds from when it does not. That is, we have not shown that there are better outcomes when high team similarity accompanies high goal similarity. As our next step, we will examine this question. If this relationship leads to positive outcomes or the lack of this relationship is associated with negative outcomes (such as delay, lower software quality, or increased maintenance cost), there is value in developing tools and practices to avoid branches with similar goals that are contributed to by disparate teams.

Further, we exhort other researchers studying various development contexts to ask the same or similar questions regarding division of work and teams using branches and report their results.

6. REFERENCES

[1] Cataldo, M., Wagstrom, P., Herbsleb, J., and Carley, K. Identification of coordination requirements: implications for the Design of collaboration and awareness tools. In *Proceedings of the 20th anniversary of the Conference on Computer supported cooperative work* (2006).

[2] Nagappan, N., Victor, B., and Brendan, M. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering* (2008).

[3] Parnas, D. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 14 (1972), 221-227.

[4] Conway, M. How do committees invent. *Datamation*, 14, 4 (1968), 28-31.

[5] Bird, C., Rigby, P., Barr, E., Hamilton, D., German, D., and Devanbu, P. The Promises and Perils of Mining Git. In *Proceedings of the Sixth Working Conference on Mining Software Repositories* (2009), IEEE Computer Society.

[6] International Working Conference on Mining Software Repositories. (2004-2010).

[7] Guimaraes, M.L. and Rito-Silva, A. Towards Real-Time Integration. In *Proceedings of the Workshop on Cooperative and Human Aspects of Software Engineering* (2010).

[8] Dourish, P. and Bellotti, V. Awareness and Coordination in Shared Workspaces. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (1992).

[9] Damian, D., Izquierda, L., Singer, J., and Kwan, I. Awareness in the Wild: Why Communication Breakdowns Occur. In *Proceedings of the International Conference on Global Software Engineering* (2007).

[10] Tan, P.-N., Steinbach, M., and Kumar, V. *Introduction to Data Mining*. Addison Wesley, 2005.

[11] Dowdy, S., Wearden, S., and Chilko, D. *Statistics for Research*. John Wiley & Sons, 2004.

[12] Conover, W.J. *Practical Nonparametric Statistics*. Wiley & Sons, New York, 1971.

[13] Benjamini, Y. and Hochberg, Y. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57, 1 (1995), 289-300.

[14] Sarma, A., Noroozi, Z., and van der Hoek, A. Palantir: Raising Awareness among Configuration Management. In *Proceeding of the 25th International Conference on Software Engineering* (2003), 444-454.

[15] Sarma, A., Redmiles, D., and van der Hoek, A. Empirical Evidence of the Benefits of Workspace Awareness in Software Configuration Management. In *Proceedings of the 16th ACM SigSoft International Symposium on Foundations of Software Engineering* (2008), 113-123.