Noname manuscript No.
(will be inserted by the editor)

Revolution or Regression?

-A Comparatively Empirical Study of Two Agile Development Processes? If so, write it here

Yujuan Jinag · Josh Chiang · Roy Budhai · Bram Adams

Received: date / Accepted: date

Abstract Insert your abstract here. Include keywords, PACS and mathematical subject classification numbers as needed.

Keywords First keyword · Second keyword · More

1 Introduction and Motivation

2 Related Work

3 Approach

3.1 Select Study Case

The dominate project P of this company S was launched in 2014 summer. At the beginning, it adopted traditional branch-based integration hierarchy. Since 2015 May, they started adopting a new mechanism-"feature toggling" in order to improve the integration and testing efficiency. This allows us to compare the two integration approaches in a rather steady context with less independent variables to control. Figure [REF] shows the two different integration approaches in this company.

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

3.2 Data Collection

We collect the data of this project from 2014 May until 2015 September, including the build information from Jenkins, reviewing information from Gerrit and development information from Git.

3.3 key Performance Indicators

We proposed a bunch of KPIs (Key Performance Indicator) to compare the two approaches from different perspectives. After discussing with engineers of this company, we nailed down the four main KPIs covering three dimensions (shown in Table 1). These KPIs are the most important ones from the industry perspective.

Integration effort indicates how difficut to integrate a commit into the main trunk, is measured by the size of merge commits. When a commit is integrated into another branch seamlessly, the size of merge commit will be zero. If a commit causes integration conflicts when being merged into another branch, developer needs to fix the conflicts by modifying integration context. The more code lines are been changed, the larger the merge commit will be. Thus, the size of merge commit can reflect the effort developers spend for integrating a new commit.

Productivity wants to measure if there is difference for development efficiency as well as the amount of developers contributions with **LOC (Line Of Code)** and **Success rate**. We sum up LOC of all commits that integrated into main trunk submitted by each developer. Success rate is used to measure how easily a commit get into main trunk. In this project, every code change made by developer needs to be submitted to Gerrit to review. If reviewers think this change needs to be revised, the developer needs to work on it and submitted a new version later. The more versions one commit goes through, the more difficult to accept new code changes. The success rate is defines as the number of commits divided by its corresponding patch revisions. For example, if a code change is accepted the first time it's reviewed, its success rate is 100%. If a code change is revised once, then its success rate is 50%.

Quality assurance is measured with the number of bug-fixes per release. In our case study, the branch stable only integrate the bug-fixes commits to stablize the staging release version, so we just need to count the number of commits integrated by stable branch. Note the bug-fixes here are pre-release bug-fixes.

4 Case Study Result

4.1 Integration Effort

Motivation: Which approach makes integration easier?

Table 1 The Four most important KPIs.

dimension	KPI
Integration Effort	number
Productivity	LOC (per developer)
	Success rate
Quality Assurance	# of bugs

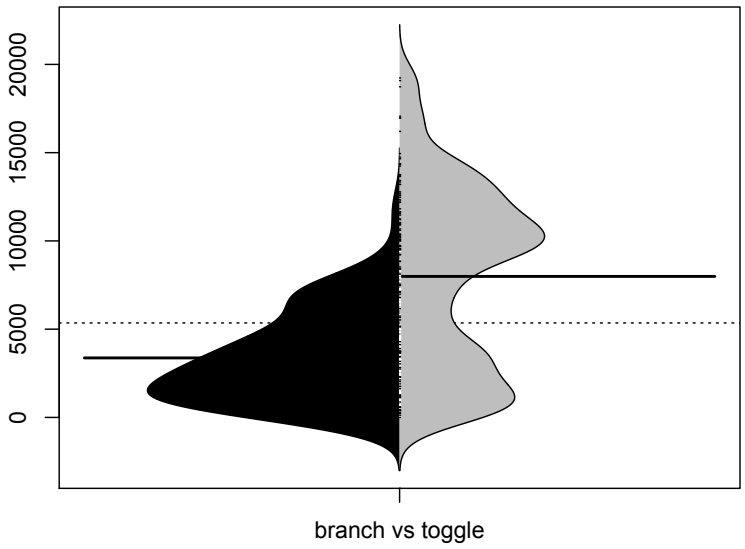


Fig. 1 Beanplot of integration effort.

Approach: Size of merge commit. Distribution across all merge commits.

Findings:

It is easier to merge branches into main trunk with branch-based approach. Figure 1 is the beanplot of merge commit size distributions for the two period of project for each merge commit. We can see the median value of all merge commits' size is much smaller than that of toggle-based approach. This means it cost less effort to merge sub-branches into main trunk in the branch-based hierarchy.

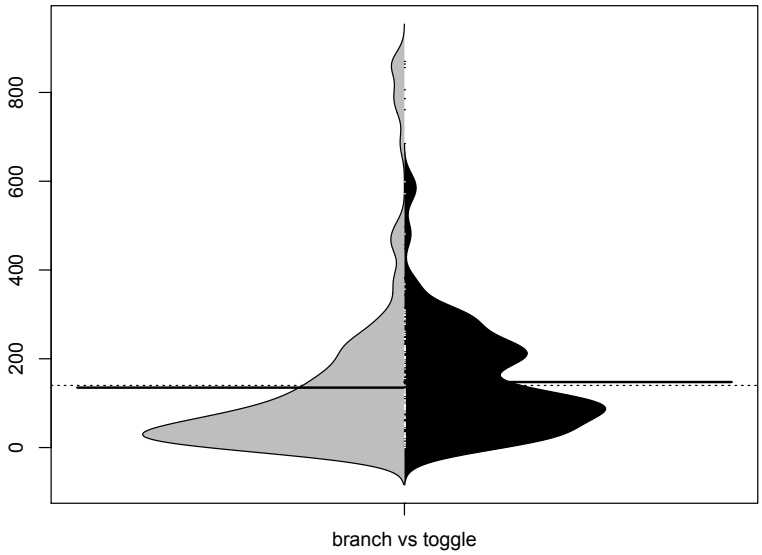


Fig. 2 Beanplot of LOC distribution per developer.

4.2 Productivity

Motivation: Which approach helps developers contribute more code and help code being qualified faster?

Approach: Sum up LOC for each developer and success rate per commit.

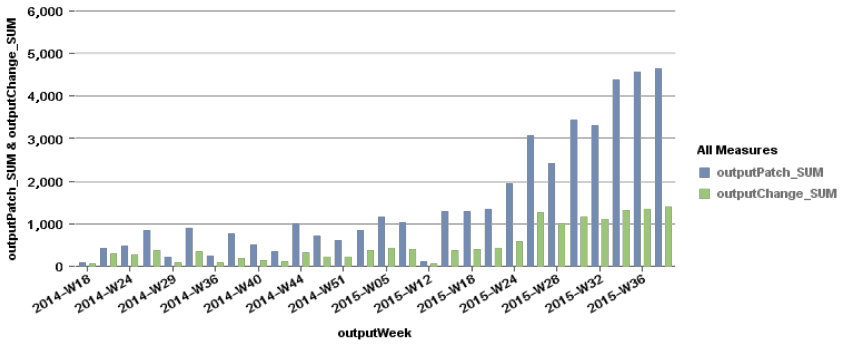
Findings:

There is no difference between two aproaches in terms of developers' contribution. Figure 2 demonstrates the distribution of LOC per developer. We can see the median value of LOC of both approaches do not differ a lot. Toggle-based approach is a bit higher.

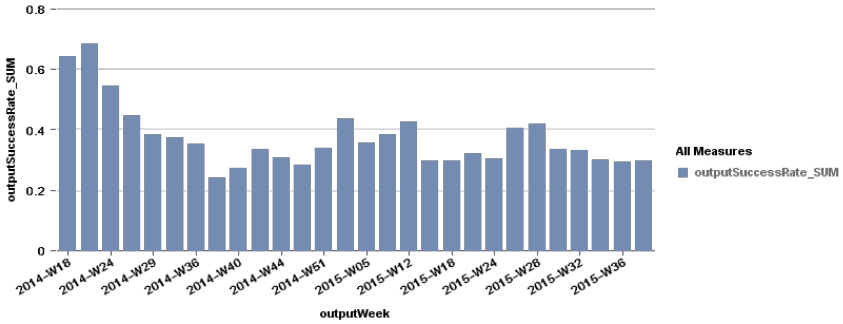
The number of commits and corresponding patch revisions are increasing while the rate is decreasing. Figure 3(a) shows the number of integrated commits and its corresponding patch revisions grouped by release. Before 2015-W18 is branch-based approach and after is toggle-based approach. We can see that the number of both increases significantly after adopting toggling mechanism. However, on the other way around the rate decreases.

4.3 Quality Assurance

Motivation: Which approach appeals more pre-release bugs?



(a) # of accepted commits and its corresponding patchset revisions



(b) Success rate rate

Fig. 3 Success rate and corresponding statistics per release.

Approach: Count number of bug-fixes per release.
Findings:

Toggle-based approach introduces more bugs. Figure 4 shows the distribution of bug-fixes for each approach per day. We found that the median value of distribution of bug-fixes for toggle-based approach is much higher than that of branch-based approach.

5 Conclusion

6 Threats to Validity

7 Acknowledgement

References

1. Author, Article title, Journal, Volume, page numbers (year)
2. Author, Book title, page numbers. Publisher, place (year)

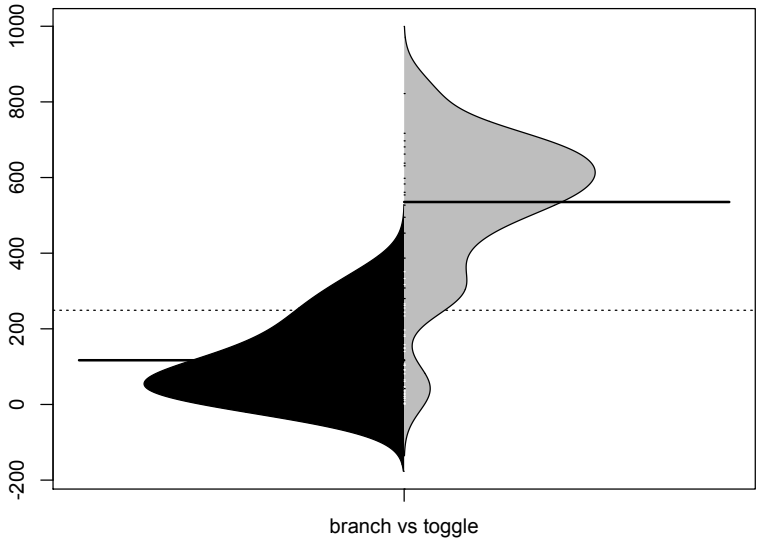


Fig. 4 Beanplot of distributions of bug-fixes.