

Cherry-Picking of Code Commits in Long-Running, Multi-release Software

Panuchart Bunyakiati Chadarat Phipathananunth

University of the Thai Chamber of Commerce

ESEC/FSE, 2017

Branch-Based Version Control Systems

- ▶ Distributed, branch-based version control systems have increasingly gained popularity in the software community.
- ▶ Github allows software to be maintained as multiple releases.
- ▶ Jetty and Python maintain multi-release branches because of the dependency constraints on external libraries.

Tartarian

- ▶ When new maintenance code in one branch is likely to be reused with some other branches.
- ▶ Tartarian can identify the branches that can benefit from reusing the code.

Bug's Life Cycle

- ▶ Code maintenance, especially bug fixing, is studied (Aranda and Venolia, 2009) [1]
- ▶ Results suggest that bug management should not consider the stage of bug (active, inactive, etc.) and workflow but should be based on satisfying “*stakeholder's goals during the lifespan of bug*” such as assignment of ownership and search for knowledges.
- ▶ Guo et al. (2010) [5] study bugs in Windows Vista and Windows 7
- ▶ Bug reporting and fixing are found to be related to individuals, social and organisational issues.

Commit Messages and Code Clones

- ▶ Several works (e.g. [3] and [4]) study the automatic generation of commit messages.
- ▶ Jiang and McMillan [6] propose a tool that automatically generate commit messages in the “*verb+object*” format.
- ▶ Our work provides a set of hashtags for developers to annotate the commit messages, which may enhance code reuse through better semi-automated communication and team communication.
- ▶ Different from code clone detection techniques [2] and [10] that analyze code to identify similarities between branches to support code reuse.

Cherry-Picking

- ▶ Cherry-picking is a process to manually pick commits in one branch and apply them into another branch.
- ▶ Developers must know which commits and which branches the commits should be applied.
- ▶ Those branches cannot be merged as they are long-running releases.
- ▶ Use *git cherry-pick* to apply the commits into another branches.

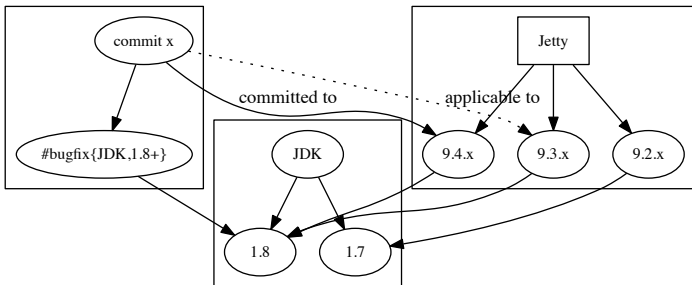
```
* b44ea88 (release1.2) Added f (cherry picked from db55fd2)
| * 4fcd797 (release1.1) Added f (cherry picked from db55fd2)
| | * db55fd2 (HEAD -> master, release1.3) Added f
| | * 3e63476 Added e
| | * f0a729e Added d
| | /
| / |
* | 9981679 Added c
* | 01ea7e6 Added b
| /
* 5597940 Added a
```

Current Practices

- ▶ Many kinds of branch-based workflows that allow cherry-picking.
- ▶ Branches can not be merged as they have to be maintained separately.
- ▶ *git cherry* only finds code diff
- ▶ *-x* appends the commit message with “*cherry picked from commit <commitid>*”
- ▶ Pull-based development involves a number of contributors who must understand all possible conflicts.

Tartarian

- ▶ identifies branches that have a potential to benefit from commits in other branches.
- ▶ helps to make decisions whether the commit should be cherry picked into a release branch.
- ▶ parses *pom.xml* and constructs a *Dependency Graph* on external libraries.
- ▶ requires developers to provide hashtags to indicate the purpose of commit regarding the dependencies.
- ▶ matches dependency graph and hashtags to identify the applicable branches for the commits.



Commit using Tartarian

- ▶ `$git tcommit -m "fixed bug 132215 #bugfix{JDK, 1.8+}"`

To check the recommendations

- ▶ *\$git tcommit -r*

Dependency Graph

- ▶ Tools such as MaX [11] can determine control and data dependencies and create a dependency graph to analyse the impact of code changes.
- ▶ Build tools such as Maven, Ant and Gradle maintain dependency configurations.
- ▶ Tartarian parses the pom.xml files and constructs a global dependency graph for all releases present.

Dependency Graph

- ▶ Dependency graph can be defined as a directed graph $G = (V, E)$
- ▶ V is a set of modules and $E \subseteq V \times V$ is a set of dependencies.
- ▶ $V(X, Y)$ indicates a module X depends on modules Y .
- ▶ “*dependency hell*”
- ▶ long chains of dependencies e.g. $V(W, X), V(X, Y), V(Y, Z)$,
- ▶ dependency conflicts e.g. $V(X, Z_{version1}), V(Y, Z_{version2})$ and
- ▶ circular dependency e.g. $V(X, Y), V(Y, Z), V(Z, X)$.

Tartarian Hashtags

- ▶ Developers describe the purpose of a commit in a commit message.
- ▶ Other developers must read the message to understand the purpose.
- ▶ If the commit message has insufficient information, it is difficult for other developers to know what he should do.

Tartarian Hashtags

- ▶ We analyse practices and guidelines, such as the JDK migration guide [12] and the deprecated list of APIs [13] and the conventional uses of cherry-picking.
- ▶ We also examine the cherry-picking practices of the Jetty project and four other open source projects including *cpython*, *elasticsearch*, *hadoop*, and *linux*.
- ▶ We analyze commit messages that contain the text “*cherry picked from*” for traceable cherry-picked commits.

Tartarian Hashtags

- ▶ For each project, we use R to find term occurrence in the messages.
- ▶ Then, use those terms as keywords that appear more frequently to classify the messages into categories.
- ▶ Based primarily on the Jetty project, we use qualitative analysis into the detail of issues and code changes, and propose the following hashtags.

Table: Tartarian hashtags

Tag	Description	Example	Priority
#bugfix	A branch will result in an error, without this commit.	#bugfix{JDK, +}	High
#backport	Backport this commit from a newer release to an older one.	#backport{Jetty, 9.2.x}	Medium
#config	Changes in configuration which may effect some of the releases.	#config{JDK, 1.8+}	High
#deprecated	A method is deprecated and there is a better alternative in this commit.	#deprecated{JDK, 1.8+}	Low
#improve	This commit provides an improvement for a higher quality of code.	#improve{Maven, +}	Low
#inaccessible	The method is found to be inaccessible, without this commit a branch will not compiled.	#inaccessible{JDK, 1.8+}	High
#removed	A method is removed but still in backwards compatibility.	#removed{JDK, 8+}	Medium

Tartarian Hashtags

- ▶ These hashtags can be prioritised into three levels of urgency
- ▶ High: must be cherry picked immediately and without them software might fail.
- ▶ Medium: must be dealt with eventually when a developer can.
- ▶ Low: the commit may be ignored but to apply it to the release can be beneficial to the quality of software.

Branch Identifier

- ▶ After a code change is committed to the repository, Tartarian checks if a commit has a relevant hashtag.
- ▶ If so, it analyses the tag and parses for the dependency constraints attached with the tag.
- ▶ The Branch Identifier represents dependency graph in Neo4j [9], a database for graph data structure, and translates the constraints into Cypher [8], a query language for graphs, to retrieve the affected release branches.

Branch Identifier

- ▶ For example, `#bugfix{JDK, 1.8+}` indicates a commit applicable for the releases that have dependency on JDK 1.8 and later.
- ▶ Each release is then checked for its dependency satisfying the constraints.
- ▶ With this information, Tartarian can notify developers who are responsible for the affected releases to consider cherry picking this commit into their branches with high priority.

Jetty 9.2, 9.3 and 9.4

- ▶ Jetty, a web server and servlet container is maintained at Github under <https://github.com/eclipse/jetty.project>.
- ▶ We use data from *jetty-9.2.x*, *jetty-9.3.x* and *jetty-9.4.x* for this study.
- ▶ To analyse cherry-picked commits in the Jetty project, we identify the commit messages that contains the phrase “*cherry picked from <commit id>*”.
- ▶ We search with the term “*cherry picked*” and “*cherry*” to trace all cherry-picked commits using the SourceTree tool.

Qualitative Analysis

- ▶ When we found such commit, we explore the intention of the commit by looking at the issue that tends to cause this commit, the full commit message and the *diff* of source code of both the original commit and the cherry-picked one.
- ▶ We use “git branch –contains <commit id>” to identify all branches that contain the commit and verify that the commit is indeed cherry picked.
- ▶ Once we have the *branch id*, we check the commit and source code in Github to analyse the changes.
- ▶ SourceTree returns 19 messages from the search with keyword “*cherry picked*” and 27 messages with keyword “*cherry*” that indicates other commits such as “*cherry pick correction*”, “*undo cherry pick*”, “*cherry pick cleanup*” etc.

#Deprecated

- ▶ It is possible that the newer version of external libraries might provide a better alternative to code.
- ▶ *-if (m.getParameterTypes().length != 0)*
- ▶ *+if (m.getParameterCount() != 0)*

source <https://github.com/eclipse/jetty.project>

#Bugfix

```
try
{
    ...
    switch(action)
    {
        ...
        case DISPATCH:
-         _request.setDispatcherType(DispatcherType.REQUEST);
-         ...
-         getServer().handle(this);
-         ...
-         break
        }
        case ASYNC_DISPATCH:
-         {
-             _request.setDispatcherType(DispatcherType.ASYNC);
-             getServer().handleAsync(this);
-             break;
-         }
        case ERROR_DISPATCH:
-         {
-             ...
-             _request.setDispatcherType(DispatcherType.ERROR);
-             ...
-             getServer().handleAsync(this);
-             break;
-         }
-         ...
-     }
-     finally
-     {
-         _request.setDispatcherType(null);
-     }
}
```

Fig. 1: The HttpChannel.java before cherry picking

```
+ try
+ {
+     _request.setDispatcherType(DispatcherType.REQUEST);
+     getServer().handle(this);
+ }
+ finally
+ {
+     _request.setDispatcherType(null);
+ }
+ ...

+ try
+ {
+     _request.setDispatcherType(DispatcherType.ASYNC);
+     getServer().handleAsync(this);
+ }
+ finally
+ {
+     _request.setDispatcherType(null);
+ }
+ ...

+ try
+ {
+     _request.setDispatcherType(DispatcherType.ERROR);
+     getServer().handleAsync(this);
+ }
+ finally
+ {
+     _request.setDispatcherType(null);
+ }
+ }
```

Fig. 2: The HttpChannel.java after cherry picking

source <https://github.com/eclipse/jetty.project>

#Configuration

```
-      <build>
-        <plugins>
-          <plugin>
-            <groupId>org.apache.maven.plugins</groupId>
-            <artifactId>maven-javadoc-plugin</artifactId>
-            <configuration>
-              <!-- Required for Java 8u121 or later -->
-              <additionalparam>
-                &#45;&#45;allow-script-in-comments
-              </additionalparam>
-            </configuration>
-          </plugin>
-        </plugins>
-      </build>
```

source <https://github.com/eclipse/jetty.project>

#Configuration

```
-      <detectJavaApiLink>true</detectJavaApiLink>
+      <detectJavaApiLink>false</detectJavaApiLink>
+      <show>protected</show>
+      <!-- needed for Java 8+ -->
+      <additionalparam>
+          &#45;&#45;allow-script-in-comments
+      </additionalparam>
```

source <https://github.com/eclipse/jetty.project>

#backport

- ▶ Backport is essential for backward compatibility and for the code in newer branch to produce the same output as those in older branches.
- ▶ In commit `5c0906e` in release 9.3.x, the commit message is *"474454 - Backport permessage-deflate from Jetty 9.3.x to 9.2.x + post cherry-pick merge cleanup"*.
- ▶ The commit message does not contain the hash of the original commit, without this information, it is very difficult to identify the original commit.

Cherry-picking in other software projects

- ▶ We found that many software projects have long-running release branches to maintain multiple versions of the software.
- ▶ Python 2.7, 3.5 and 3.6, the Python community has to maintain both versions for several years.
- ▶ Elasticsearch and Hadoop maintain a number of release branches.
- ▶ For Linux, one master development branch, 513 releases as of June 2017.

Cherry-picking in other software projects

- ▶ To generalise the cases found in the Jetty project to other open source projects.
- ▶ Based on this high-level analysis, the cherry-picked commits can be categorized into three groups.
- ▶ “*bug fixes*” for commits related to bugs,
- ▶ “*backports*” for commits regarding backward compatibility,
- ▶ “*code maintenance*” for the messages that contain verb words such as ‘*add*’, ‘*remove*’, ‘*improve*’, ‘*change*’ etc.

Table: Cherry-picks in four major open source projects

Project	release branches	total	bug fixes	back-ports	code maintenance
cpython (python/cpython)	2.7, 3.5, 3.6	461	149 (32.32%)	10 (2.17%)	264 (57.27%)
elasticsearch (elastic/elasticsearch)	2.0-2.4, 5.0-5.4, 5.x	337	49 (14.54%)	1 (0.30%)	269 (79.82%)
hadoop (apache/hadoop)	2, 2.6-2.8.1, 3.0.0-alpha1, 2 and 3	7150	1431 (20.01%)	-	2095 (29.30%)
linux (torvalds/linux)	-	534	98 (18.35%)	-	119 (25.81%)

Conclusion and Future Work

- ▶ We propose that, together with commit messages, developers may use hashtags as metadata to help specifying the intention of a commit.
- ▶ These hashtags, when used with the Tartarian tool, can identify the release branches that a commit may be reused.
- ▶ Doing so can help developers to efficiently maintain software without the need to acquire implicit knowledge of the release branches.
- ▶ Despite the fact that dependencies on JDK are used in most of the case studies in this paper, we believe that Tartarian can be applied to dependency constraints in general, which we will explore in our future work.

Limitations

- ▶ The satisfaction of dependency constraints alone is not sufficient for the cherry-picked commit to fit the branch.
- ▶ Cherry-picking requires semantic checking to establish that the cherry-picked commit will not cause unintentional faults or build breaks in the branch.
- ▶ A feature that checks for these semantic constraints is needed in Tartarian.

Acknowledgement

- ▶ The authors are grateful to the reviewers for their helpful suggestions and would like to thank the University of the Thai Chamber of Commerce for supporting this project.

Bibliography I



Jorge Aranda and Gina Venolia. 2009. The secret life of bugs: Going past the errors and omissions in software repositories. In Proceedings of the 31st International Conference on Software Engineering (ICSE '09). IEEE Computer Society, Washington, DC, USA, 298-308.






Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. 2007. Comparison and evaluation of clone detection tools. IEEE Transactions on software engineering, Vol. 33, no. 9. IEEE.



Casalnuovo, Casey. Analyzing and Generating Commit Messages for Software Repositories. Diss. University of Delaware, 2013.




Bibliography II

-  Corts-Coy, Luis Fernando, et al. 2014. On automatically generating commit messages via summarization of source code changes. In Proceedings of the 14th International Working Conference on Source Code Analysis and Manipulation (SCAM '14). IEEE, USA, 275-284.
-  Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE '10), Vol. 1. ACM, New York, NY, USA, 495-504.
-  Siyuan Jiang and Collin McMillan. 2017. Towards automatic generation of short summaries of commits. In Proceedings of the 25th International Conference on Program Comprehension (ICPC '17). IEEE Press, Piscataway, NJ, USA, 320-323.

Bibliography III

-  Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M. German. 2015. Open source-style collaborative development practices in commercial projects using GitHub. In Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15), Vol. 1. IEEE Press, Piscataway, NJ, USA, 574-585.
-  Neo Technology. Cypher Introduction. <http://neo4j.com/docs/developer-manual/current/cypher/#cypher-intro>. Retrieved July, 2017.
-  Neo Technology. Neo4J. <http://neo4j.com>. Retrieved July, 2017.
-  Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Sci. Comput. Program. 74, 7 (May 2009), 470-495.

Bibliography IV

-  Amitabh Srivastava, Jay Thiagarajan, and Craig Schertz. 2005. Efficient integration testing using dependency analysis. Technical Report MSR-TR-2005-94, Vol. 82. Microsoft Research.
-  Oracle. Java Platform, Standard Edition Oracle JDK 9 Migration Guide. <https://docs.oracle.com/javase/9/migrate/>. Retrieved June, 2017.
-  Oracle. Deprecated API. <https://docs.oracle.com/javase/8/docs/api/deprecated-list.html>. Retrieved June, 2017.