Social Coding Platforms Facilitate Variant Forks
(Keynote REVE-WEESR 2022)

Prof. Serge Demeyer
AnSyMo

Universiteit Antwerpen
https://secoassist.github.io/
Collaborators
Variant Forks – Motivations and Impediments

John Businge, Ahmed Zerouali, Alexandre Decan, Tom Mens, Serge Demestres, and Cees De Rosis

University of Antwerp, Antwerp, Belgium

abstract

Software and its engineering have evolved, which can be used as-is to support the management of divergent variants.

CC Concepts

Software and its engineering – Software version control

Software maintenance – Software architecture

1 Introduction

Code reuse is the practice of reusing existing code to speed up the development process. ‘Divergent’ code reuse is motivated by the existence of software artefacts that can be used instead of new code to achieve the same result.

PaReco: Patched Clones and Missed Patches among the Divergent Variants of a Software Family

Poilevdene Radjel Ramkisoen, John Businge, Joree van Bradel, Alexandre Decan, Serge Demestres, Cees De Rosis, and Frédo Khandekar

University of Antwerp & Flanders Make, Belgium

1 INTRODUCTION

Software and its engineering – Software version control

Software maintenance – Software architecture

1 INTRODUCTION

Code reuse is the practice of reusing existing code to speed up the development process. ‘Divergent’ code reuse is motivated by the existence of software artefacts that can be used instead of new code to achieve the same result.
November 2021

73M+
Total developers
on GitHub

170M+
Pull requests
merged

200M+
Repositories

28M+
Public
Repositories
The Equifax data breach occurred between **May and July 2017** at the American credit bureau Equifax. Private records of 147.9 million Americans along with 15.2 million British citizens and about 19,000 Canadian citizens were compromised in the breach, making it one of the largest cybercrimes related to identity theft.

Wired Magazine, “Equifax has no excuse”, September 2017

---

**EQUIFAX**

**DATA BREACH**

May 2017

---

**CVE-2017-5638**

March 2017

---

**Failure to patch two-month-old bug led to massive Equifax breach**

Critical Apache Struts bug was fixed in March. In May, it bit ~143 million US consumers.
1. Social Forks

Social Fork

Original project

Social Fork

2. Variant Forks

Social Fork

Original project

Variant 1

Variant 2

Social Fork

Focus

Optionally

Pull request

VS
Open-source keyboard firmware for Atmel AVR and Arm USB families


- ZSA, forked for QMK Oryx Configurator (to safeguard stability)
- gboards.ca keyboard firmware is maintained separately from qmk/master
Open Science

1. Programming language: Java, Python, C, PHP, ...
2. Dedicated projects: Android, Blockchain, Eclipse, ...

Variant | Upstream
--- | ---
Social? | Fork 1
Variant | Fork 2
Social? | Fork 3
Social? | Fork 4
Variant | Fork 5

Software family
upstream
Fork2
Fork5

Java

12
Variant forks – Motivations and Impediments

John Businge,² Alexandre Zerouali,¹ Alexandre Decan,¹ Tom Mens,¹ Serge Demeyer,¹ and Coen De Rovers²

¹ Universiteit Antwerpen, Antwerp, Belgium
² Campus Singapore, Nanyang Technological University, Singapore

Abstract

In this paper, we study the evolution of variant forks in the context of open source software (OSS) projects. Variant forks are created to address diverging requirements or to reuse code of other projects. We conducted an exploratory qualitative study on 105 maintainers of variant forks to understand the motivations behind creating and maintainingvariant forks, as well as the impediments. Our findings indicate that variant forks are often created to address divergence in requirements or to reuse code of other projects. However, maintaining variant forks is challenging due to issues such as code integration, maintenance, and coordination among maintainers. The results of this study can guide future empirical studies and inform the design of tools to support the management of variant forks.

1. Introduction

Variant forks are common in the context of OSS projects. Variant forks are created to address diverging requirements or to reuse code of other projects. The evolution of variant forks is driven by various motivations such as code reuse, maintenance, and coordination among maintainers. To better understand the motivations and impediments behind creating and maintaining variant forks, we conducted a qualitative study on 105 maintainers of variant forks.

2. Motivations

Variant forks are created to address various motivations such as code reuse, maintenance, and coordination among maintainers. The motivations behind creating variant forks are driven by the need to address divergence in requirements or to reuse code of other projects. The motivations behind maintaining variant forks are driven by the need to maintain the co-existing projects.

3. Impediments

Maintaining variant forks is challenging due to various impediments such as code integration, maintenance, and coordination among maintainers. The impediments behind creating and maintaining variant forks are driven by the need to address various challenges such as code integration, maintenance, and coordination among maintainers.

4. Conclusion

Our findings indicate that variant forks are often created to address divergence in requirements or to reuse code of other projects. However, maintaining variant forks is challenging due to issues such as code integration, maintenance, and coordination among maintainers. The results of this study can guide future empirical studies and inform the design of tools to support the management of variant forks.

References


All of these studies and many others were conducted in the pre-GitHub days. Current GitHub days only two studies. Little is known about the motivations of creating variants on social coding platforms.
We designed a 12-question survey that included both closed and open-ended questions.
Any original developer in variant?  
Any active common maintainer between variants?  
Decision to create variant?  
Motivation for creating variant?  
Motivation detail?

Confirmed previous findings

New findings

- maintenance
- different goals
- new features
- customization
- unmaintained feature
- technology
- enhancement
- new release
- stripped version
- responsiveness
- feature acceptance
- differences
- supporting personal projects
- supporting upstream
- code quality
- community related
- localization
- upstream lacks the resource
- closed source
Previous studies identified four categories of motivations for creating variant forks:
- technical (e.g., diverging features),
- governance (e.g., diverging interests),
- legal (e.g., diverging licenses), and
- personal (e.g., diverging principles).

Do they still hold?
[R36]. Motivation – legal. “The founders of the mainline had been absent from the project for several years, but came back and booted the maintainers off and […] shifted the project to a closed source.”

Motivation detail – closed source.

[R59]. Motivation – governance. “The PR to merge the fork's new capabilities into the mainline code was too large, […] and my attempts to incorporate feedback into the PR […] ended upsetting the primary maintainer who has been studiously ignoring the pull request for three years.”

Motivation detail – responsiveness.

[R18], motivation – others. “[The] maintainer was not interested in a PR that added functionality needed by a project I’m developing. [It] was considerably easier to add the logic into the [new] library than bolt it on”.

Motivation-detail – supporting personal projects.
How do variant projects evolve with respect to the mainline?

SQ. Do the variant forks and the original project still discuss the main directions of the project?

never: 53 (91.3%)
yes: 5 (9.2%)

[R67]. “changes are out of scope.”
[R36]. “mainline is hostile to variant.”
[R57]. “We used to discuss but not anymore since the projects have technically diverged”
[R54]. “Made PRs with changes but those have just been ignored. They’re still "open" with 0 comments from the mainline dev”

How do variant projects evolve with respect to the mainline?
How do variant projects evolve with respect to the mainline?

SQ. How often do the maintainers of the variant integrate the following types of changes to and from the mainline?

- New features
- Bug fixes
- Security fixes
- Refactorings
- Documentation
- Others

Reasons for lack of interaction (Impediments):

i. technical divergence
ii. governance disputes
iii. diverging licenses
iv. distinct development teams
Variant forks – motivations and impediments.
Proceedings SANER 2022

PaReco: Patched Clones and Missed Patches among the Divergent Variants of a Software Family

Postdoctoral Radjel Ramineh, John Businge, Brent van Brakel, Alexandre Decan, Serge Demeyer, Corin De Roover, and Frédo Krichel
p.reco@vub.be, radjel.ramineh@uantwerpen.be, serge.demeyer@uantwerpen.be, andre.decan@umons.ac.be

ABSTRACT

We extend our current exploratory study on why developers create and maintain variant forks to support an analysis of the motivations and impediments. For this, we have surveyed 380 software practitioners, asking them why they forked a repository and what impediments they encountered. Next, we performed a text mining analysis to uncover concrete reasons for this phenomenon. We discuss our findings with a special focus on three questions: (RQ1) Why do developers create and maintain variant forks? (RQ2) How do variant projects evolve with respect to the mainline project? (RQ3) What are the main impediments that developers face with respect to the variant project?

The analysis of the survey responses shows that developers create variant forks either for the goal of sharing code (74.6%) or for development (61.3%). The most common impediments are due to the divergent nature of the variant project (86.8%) and difficulty for tracking changes (79.7%).}

 Patched clones and missed patches among the divergent variants of a software family.
Proceedings ESEC/FSE 2022

SECO-ASSIST
Problem
Concrete Example: Missed Opportunity

Buggy code from upstream

```c
1  return;
2  }
3  
4  while (p < (uint16_t*)SYMVAL(__eprom_workarea_end__));
5  
6  flashend = (uint32_t)((uint16_t*)SYMVAL(__eprom_workarea_end__) - 1);
```

Patched code from upstream

```c
1  return;
2  }
3  
4  while (p < (uint16_t*)SYMVAL(__eprom_workarea_end__));
5  
6  flashend = (uint32_t)(p - 1);
```

Diff for patch in upstream

```diff
@@ -363,7 +363,7 @@
1  
2  
3  -   while (p < (uint16_t*)SYMVAL(__eprom_workarea_end__));
4  +   flashend = (uint32_t)((uint16_t*)SYMVAL(__eprom_workarea_end__) - 1);
5  +   flashend = (uint32_t)(p - 1);
```

File from divergent fork at git_head

```c
1  return;
2  }
3  
4  while (p < (uint16_t*)SYMVAL(__eprom_workarea_end__));
5  
6  flashend = (uint32_t)((uint16_t*)SYMVAL(__eprom_workarea_end__) - 1);
```
Concrete Example: Effort Duplication

Buggy code from upstream

```
1   # http://ss64.com/nt/syntax-esc.html
2     _escape_re = re.compile(r'(?!<!\")(&<>)[(?<!\")\"(?![&<>])\']')
3     _escaper = partial(_escape_re.sub, lambda m: m.group(0))
```

Patched code from upstream

```
1   # http://ss64.com/nt/syntax-esc.html
2     _escape_re = re.compile(r'(?!<!\")(&<>)[(?<!\")\"(?![&<>])\']')
3     _escaper = partial(_escape_re.sub, lambda m: m.group(0))
```

Diff for patch in upstream

```
@@ -24,7 +24,7 @@
  3     _escape_re = re.compile(r'(?!<!\")(&<>)[(?<!\")\"(?![&<>])\']')
  4     _escape_re = re.compile(r'(?!<!\")(&<>)[(?<!\")\"(?![&<>])\']')
  5 +    _escaper = partial(_escape_re.sub, lambda m: m.group(0))
  6     _escaper = partial(_escape_re.sub, lambda m: m.group(0))
```

File from divergent fork at `git_head`

```
1   # http://ss64.com/nt/syntax-esc.html
2     _escape_re = re.compile(r'(?!<!\")(&<>)[(?<!\")\"(?![&<>])\']')
3     _escaper = partial(_escape_re.sub, lambda m: m.group(0))
```
MO – Missed opportunity  
ED – Effort duplication  
SP – Both buggy and patched lines  
NI – Uninteresting  
CC – Unhandled programming language  
NE – Missing file in target  
EE – Error
8,323 patches from 364 source variants

2,225 interesting patches

MO – Missed opportunity
ED – Effort duplication
SP – Both buggy and patched lines
NI – Not interesting
CC – Unhandled programming language
NE – Missing file in target
EE – Error

<table>
<thead>
<tr>
<th>Classification</th>
<th>Number of Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO</td>
<td>1,008</td>
</tr>
<tr>
<td>ED</td>
<td>1,116</td>
</tr>
<tr>
<td>SP</td>
<td>101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classification</th>
<th>Number of Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>NI</td>
<td>1,116</td>
</tr>
<tr>
<td>CC</td>
<td>101</td>
</tr>
<tr>
<td>NE</td>
<td>101</td>
</tr>
<tr>
<td>EE</td>
<td>1,008</td>
</tr>
</tbody>
</table>

Precision: 91.0%
Recall: 80.2%
Accuracy: 88.0%
F1-Score: 85.3%
LinkedIn is a clone-and-own variant of Apache Kafka that was created by copying and adapting the existing code of Apache Kafka.

...  
LinkedIn has 500 individual commits, and Apache Kafka has 3,103 individual commits.

...  
Your assignment is to identify numerous patches from patches.xls that are of different sizes and integrate them in the source variant LinkedIn. The size can be measured in terms of number of commits, files_changed, added_lines, deleted_lines, modules.
Cherry Picking – Merge Conflicts