How do Developers Fix Cross-project Correlated Bugs?

A case study on the GitHub scientific Python ecosystem

Wanwangying Ma, Lin Chen, Xiangyu Zhang, Yuming Zhou, Baowen Xu
Nanjing University, China and Purdue University, USA
Contents

- Motivation
- Objective
- Study Design
- Study Results
- Discussions
- Conclusion
Dependency between projects

Astropy

SciPy

IPython

Numpy

A great number of projects

7 other projects
Motivation—Cross-project correlated bugs

An average of 17.28% of bugs are cross-project ones.
Motivation—Cross-project correlated bugs

Part 1

Upstream project: numpy

- Related with NumPy

Downstream project: astropy

- Sending a test

Timeline:

1. Related with NumPy
2. Finding correlated issue
3. Sending a test
4. Fixing and testing

- Closed

References:
- numpy/numpy#6467
- astropy/astropy#4259
Motivation—Cross-project bugs

Survey results

- Compared with within-project bugs,
  - DQ6. more difficult to deal with?
  - DQ3. have more severe impact?
  - UQ4. pay more attention?

DQ: for downstream developers  UQ: for upstream developers

Statistical comparison

- Cross-project bugs vs. within-project bugs
- Based on the data collected from bug reports

Results:
- Requiring more time to fix
- More comments in bug reports
- More participants during fixing

- More severe impact
- More difficult to fix
- Attracting more attention
Objective

To investigate how software practitioners fix cross-project correlated bugs

Focusing on two aspects:

1. cross-project root cause tracking
   - as the bug carries over from one project to another, it becomes harder to trace the bug back to its root

2. coordination in bug fixing
   - while waiting for an upstream fix, the downstream developers need to coordinate their project with the upstream one in order to minimize any undesirable impact of the cross-project bugs
Study design—*Studied Projects*

- **GitHub Scientific Python ecosystem**
  - Seven seed projects
    - IPython
      - Enhanced Interactive Console
    - NumPy
      - Base N-dimensional array package
    - Matplotlib
      - Comprehensive 2D Plotting
    - SciPy library
      - Fundamental library for scientific computing
    - pandas
      - Data structures & analysis
    - astropy
      - A Community Python Library for Astronomy

- Totally 271 pairs of cross-project correlated bugs
- Involving 204 projects
Study design—Research Questions

Research questions

1. How long does it take to find the root cause of cross-project correlated bugs, that is, to link the downstream bug to the criminal upstream bug?

2. What factors are important to track the root cause of cross-project correlated bugs?

3. How do downstream developers coordinate with upstream projects to deal with cross-project correlated bugs after identifying the root cause?
Study design—Research Methods

- Manual inspection
  - Three authors of the paper

- Online survey
  - 116 responses
  - Response rate: 17.2%

Summarizing the findings
Results - RQ1: Difficulty of Finding the Root Cause

- Manual inspection
  - How long to find the root cause?

The root causes of nearly half of the cross-project bugs are identified in a relatively long time (one day to more than 100 days).

Part 4
Survey results

DQ2: is it difficult to find the root causes for cross-project bugs?

- 60.5% of the downstream developers thought it difficult or very difficult to find the root cause.
Results - RQ2: Factors for Tracking the Root Cause

- Manual inspection
- Stack traces: the sequences of calls to the failure
- Communication: between upstream and downstream developers
- Familiarity: expertise in the buggy component
Survey results

DQ4: what factors may act as positive roles to find the root-causes of cross-project bugs?

- Stack traces: 73.8%
- Communication: 72.6%
- Familiarity: 63.1%
- Others: 9.5%
- Others: test cases, documentation, stack overflow, ...

Part 4
## Communication

**Attitude**

"One is rarely facile with the upstream project's internals, so communication is essential"

**Focus**

Responsiveness: early and friendly responses

**Results - RQ2: Factors for Tracking the Root Cause**

<table>
<thead>
<tr>
<th>Downstream</th>
<th>Upstream</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attitude</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;One is rarely facile with the upstream project's internals, so communication is essential&quot;</td>
<td>UQ3. As an upstream developer, do you care about the opinions from the downstream projects or communicate with the downstream developers?</td>
</tr>
<tr>
<td><strong>Focus</strong></td>
<td></td>
</tr>
<tr>
<td>Responsiveness: early and friendly responses</td>
<td>Content: concrete description of the bug and the requirements of the downstream project</td>
</tr>
</tbody>
</table>

![Bar Chart](chart.png)
Results - RQ3: Practices of Downstream Developers

- Manual inspection

- Working around the bug locally (60)
  Workaround: a temporary solution injected in the downstream code locally

- Restricting the dependent upstream versions (8)

- Doing nothing bug waiting for the upstream fix (49)
Survey results

DQ7. What do downstream developers usually do with a cross-project bug?

- **A.** Proposing a workaround
- **B.** Restricting the upstream versions
- **C.** Doing nothing but waiting
- **D.** Using a different upstream project
- **Others:** Actively help the upstream project by proposing/pushing solutions

"Whatever is easiest in their specific circumstances, above are good examples! but probably work around the issue."
Results - RQ3: Practices of Downstream Developers

- **Workaround**
  - **Problems:**
    - version-dependent codes
    - adding maintenance burden
  - **Implications:**
    - tools to support synthesis and maintenance of workarounds

A bug in *numpy* 1.6 affected *astropy*.

- **Affected code in astropy:**
  ```python
  format_ufunc = np.vectorize(do_format, otypes=['U'])
  result = format_ufunc(values)
  ```

- **Workaround:**
  ```python
  if numpy_version < 1.7:
      # work around it
      new code
  else:
      ```
Discussions—Dilemmas in collaboration

Cross-project testing
→ to prevent cross-project bugs

- **Downstream**
  - it would be helpful if the testing suites for downstream projects are run before releasing an upstream version.

- **Upstream**
  - impossible to get the complete list of downstream projects
  - different ways for downstream projects to run their tests
  - time consuming

- to develop tools for effective cross-project testing
Notifications of bug fixes

- To deprecate outdated workarounds

- **Downstream**
  - Helpful

- **Upstream**
  - An extra burden

DQ8. Is it necessary for the affected downstream projects to be notified?

- Yes: 69.1%
- No: 30.9%

Discuss the necessity of notifying affected downstream projects to improve the notification scheme of GitHub so that it can send automatic massages.
Releasing the bug fix version

Problem: “release cycles of downstream and upstream projects are out of sync”

- **Downstream**
  - hoping for a quick release

- **Upstream**
  - preferring to give a bit of time for reflection

UQ8. When scheduling a bug-fix release, will you consider the requirements of the important downstream projects?

![Bar chart](chart.png)

“The reformation should help best of both-ends.”
Related Work

 Practices in fixing bugs

T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened"

G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta, "Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD"

S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users"

 Collaboration on GitHub

L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: transparency and collaboration in an open software repository"

A. Lima, L. Rossi, and M. Musolesi, "Coding together at scale: GitHub as a collaborative social network"

 Evolution of software ecosystems

J. Bosch and P. M. Bosch-Sijtsema, "Softwares product lines, global development and ecosystems: collaboration in software engineering"

A. Decan, T. Mens, M. Claes, and P. Grosjean, "On the development and distribution of R packages: an empirical analysis of the R ecosystem"
Conclusion and Future Work

- How do developers fix cross-project bugs?
  - More difficult to repair and more severe
  - Beneficial factors for finding the root cause
    - Stack traces, communication, and familiarity
  - Common practices for downstream developers
    - The workaround

- Future work:
  - Workarounds
  - Tool support
Thank you!

Q & A

Motivation—Cross-project bugs

- Survey results
- Statistical comparison

Results—RQ1: Difficulty of Finding the Root Cause

- Manual inspection
- Higher deviation
- Time to find root cause

Results—RQ2: Factors for Tracking the Root Cause

Manual inspection

- Stack traces
- Communication
- Familiarity

Results—RQ3: Practices of Downstream Developers

Survey results

- Workaround
- Documentation
- Code injection
- Verification and testing
- Documentation and feedback

Results—RQ3: Practices of Downstream Developers

Implications
- Conditional checks to support northbound and southbound interoperability
- Better process for managing test cases and bug states
- Improved communication between developers and testers
- Enhanced use of version control systems

Fig. A summary of the workflow in the file structure/summary/regular.py