Large-Scale Patch Recommendation at Alibaba

Xindong Zhang[1], Chenguang Zhu[2], Yi Li[3], Jianmei Guo[1], Lihua Liu[1], and Haobo Gu[1]
1. Alibaba Group 2. University of Texas at Austin 3. Nanyang Technological University

Motivation

50% time On average, 49.9% of software developers' time has been spent in debugging [1]

50% cost Half of the development costs are associated with debugging and patching [2]

Automated patch recommendation can significantly reduce developers' debugging efforts and the overall development costs

Challenges

1. Accurate patch mining is difficult
- Insufficient documentation
- Manually labeling bug-fixes is hardly possible

2. Test cases are not always available
- Induce difficulty on patch validation

3. Practical requirements
- Used in everyday development routine

4. Applications are diverse
- Domain-specific techniques are not suitable, need a general approach

Our Solution

1. Does not rely on labeled defects/patches
- Automatically mines templates from historical changes

2. Does not rely on test cases to validate patches
- Allow developers to provide feedback on quality

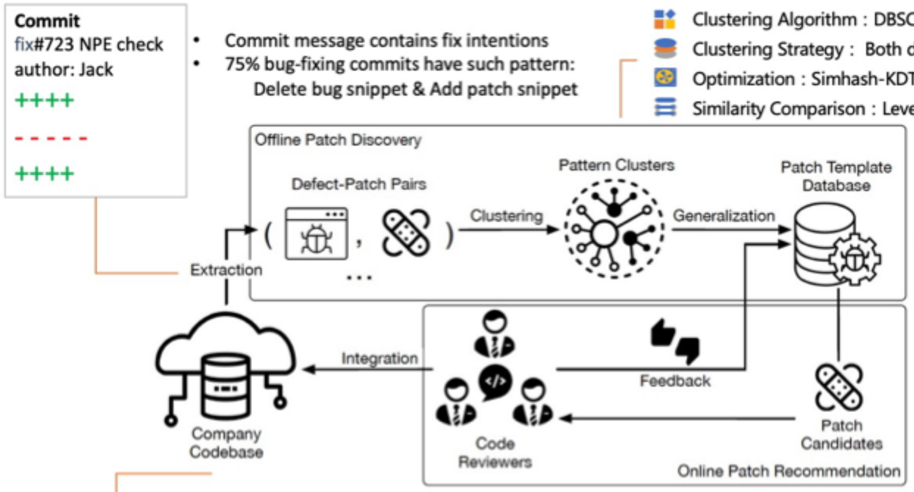
3. Guarantee high responsiveness and low FPR
- Separate patch discovery with patch recommendation
- Recommend patch within milliseconds

4. Can handle diverse applications
- Patches are mined from the entire internal codebase, using generic features

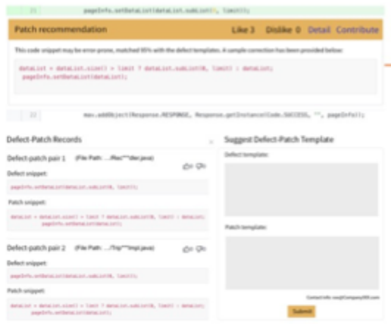
PRECFIX: Large-Scale Patch Recommendation by Mining Defect-Patch Pairs

- Commit message contains fix intentions
- 75% bug-fixing commits have such pattern: Delete bug snippet & Add patch snippet

Clustering Algorithm : DBSCAN
Clustering Strategy : Both defect & patch snippets
Optimization : Simhash-KDTree, API sequence
Similarity Comparison : Levenstein + Jaccard

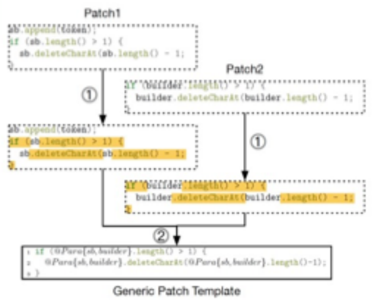


15 million commits
30 million files



This code snippet may be error-prone, matched 95% with the defect templates. A sample correction has been provided below:

```
dataList = dataList.size() > limit ? dataList.subList(0, limit) : dataList;
pageInfo.setDataList(dataList);
```



```
sb.append(token);
if (sb.length() > 1) {
  sb.deleteCharAt(sb.length() - 1;
}
```

```
if (builder.length() > 1) {
  builder.deleteCharAt(builder.length() - 1;
}
```

```
if (@Para{sb,builder}.length() > 1) {
  @Para{sb,builder}.deleteCharAt(@Para{sb,builder}.length()-1);
}
```

Results

EFFECTIVENESS — **22%**
False positive rate is 22% in patch discovery and it is supposed to be gradually reduced by feedback on discovered patch and contribution of new patch

USER STUDY — **10/12**
The majority (10/12) of the interviewed developers acknowledged the value of the patches, and all of them would like to see Precfix adopted in practice

EFFICIENCY — **5 Hours**
Offline patch discovery costs 5 hours (extracting pairs, clustering, and extracting templates consumes 22, 270, and 5 min). Online patch recommendation is made within milliseconds

DEPLOYMENT — **1 Year**
Precfix has been deployed in Alibaba for about one year so far. Every week, it recommends about 400 patches to developers on average, and receives about two to three false positive reports

Patch Categories

Randomly sampled 50 patches and made classification.

API Modification (40%)

```
  multipleSource.setParams(
    MultiSourceConvertUtil.buildReqParams(
-     itemSku.getItemId().getValue(),
+             itemSku.getConfigId(),
              itemSku.getSkuId(),
              itemSku.getSellerId(),
+             itemSku.getGpuId()},
              multipleSource.getPageSize(),
              multipleSource.getPageIndex()));
```

Validation Check (26%)

```
+ if (accountStatus != null && accountStatus.length > 0) {
      query.addCondition(new In(STR, accountStatus));
+ }
```

API Wrap (14%)

```
- String ip = host.getIp();
- String url ="http://".concat(ip).concat(flowHost);
- HttpResponse<String> response = httpRequest.asString();
- ErrorUtil.checkError(httpRequest, response, TREND, start);
- bodys.add(response.getBody());
+ String url = UrlUtil.getUrl(headHost, flowHost);
+ UrlUtil.requestAndCheckThenFillResult(httpRequest, bodys, TREND, st
```

References

[1] T. Britton, L. Jeng, G. Carver, and P. Cheak, "Reversible debugging software - quantify the time and cost saved using reversible debuggers," 2013.
[2] Britton et al. Quantify the time and cost saved using reversible debuggers. Cambridge report, 2013.