Recurring Bug Fixes in Object-Oriented Programs

- Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar Al-Kofahi, Tien N. Nguyen

Presenter

- Wasique Islam Shafin (40304330)

Introduction & Motivation

Previous studies confirms existence of recurring bug fixes where a bug-fixing change is recurring if repeated identically or with slight modifications across multiple code fragments or revisions.

But it created more questions

- Why, where and how often do these changes reoccur?
- How can they be characterized and recognized?
- How to help developers fix them effectively?

Dataset

The dataset comes from seven experienced programmers who manually reviewed bug fixes in around a thousand fixing revisions across five popular open-source projects with thousands of fixing changes.

Found high concentration recurring fixes in code peers

Project	App. Type	Revision Range	Fixes
ArgoUML	Graphic Modeling	2 - 1130	2318
Columba	Mail Client	4 - 370	829
ZK	Ajax Framework	2400 - 6200	490
FlashRecruit	Job Listings	100 - 600	1007
gEclipse	Dev Environment	400 - 10300	1126

Table 1: Subject Systems

Project	RBF	Percentage	In Space	In Time	Both
ArgoUML	390	16.8%	96.9%	17.2%	14.1%
Columba	377	45.4%	88.8%	17.7%	6.5%
ZK	188	38.4%	91.5%	13.3%	4.8%
FlashRecruit	244	24.2%	85.3%	22.1%	7.4%
gEclipse	215	19.1%	89.1%	27.7%	16.8%

Table 2: Manually Identified Recurring Fixes

Methodology I

1. Identify Code Peers in OOP
2. Recognize Recurring Bug Fixes
3. Recommend Fixing Changes from historical bug fixes

code peers are the code units (e.g. methods, classes) having similar object interactions

Finds Peer candidates for recommendations

Example Code Peer

```
public void setColspan(int colspan) throws WrongValueException{
 if (colspan <= 0) throw new WrongValueException(...);
 if (_colspan != colspan) {
  _colspan = colspan;
  final Execution exec = Executions.getCurrent();
  if (exec != null && exec.isExplorer()) invalidate() ;
  smartUpdate("colspan", Integer.toString(_colspan));...
```

```
public void setRowspan(int rowspan) throws WrongValueException{
 if (rowspan <= 0) throw new WrongValueException(...);
 if (_rowspan != rowspan) {
  _rowspan = rowspan;
  final Execution exec = Executions.getCurrent();
  if (exec != null && exec.isExplorer()) invalidate();
  smartUpdate("rowspan", Integer.toString(_rowspan));...
```

Figure 1: Bug Fixes at v5088-v5089 in ZK

```
public class UMLOperationsListModel extends
        UMLModelElementCachedListModel{
public void add( int index){
 Object target=getTarget();
 if (target instanceof MClassifier) {
  MClassifier classifier=(MClassifier)target;
  Collection oldFeatures=classifier.getFeatures();
  MOperation newOp=MMUtil.SINGLETON.buildOperation(classifier);
  classifier.setFeatures(addElement(oldFeatures,index,newOp,
          _operations.isEmpty()?null: _operations.get(index)));
```

```
public class UMLAttributesListModel extends
        UMLModelElementCachedListModel{
public void add( int index){
 Object target=getTarget();
 if (target instanceof MClassifier) {
  MClassifier classifier=(MClassifier)target;
  Collection oldFeatures=classifier.getFeatures();
  MAttribute newAt=MMUtil.SINGLETON.buildAttribute(classifier);
  classifier.setFeatures(addElement(oldFeatures,index,newAt,
          _attributes.isEmpty()?null: _attributes.get(index)));
```

Figure 3: Bug Fixes at v0459-v0460 in ArgoUML

Methodology II

```
IdentifyCodePeer(Prog)
 M.add(SimilarMethod(Prog)) //add cloned methods as candidates
 C.add(SimilarClass(Prog)) //find similar classes and
 C.add(SimilarFixedClass(Prog)) //classes with recurring fixes
 M.add(SimilarNamedMethod(C)) //match methods as candidates
 do
   (A.m, B.n) = M.next() //repeatedly process candidates
   if Sim(U_I(A.m), U_I(B.n)) ≥ σ_1 or
         Sim(U_E(A.m), U_E(B.n)) ≥ σ_2 //if similar enough
     move (A.m, B.n) from M to P_M //add as peers
     C.add((A, B)) //and check enclosing classes
     M.add(SimilarNamedMethod((A, B))) // for new candidates
 while new peers are still identified
 P_C.add(PeerClass(C)) //find peer classes
```

Figure 8: Code Peer Identification

Methodology III

```
RecognizeRecurringFixes(Fixes)
 for each Δ ∈ Fixes
    IU(Δ) = ImpactUsage(Δ) //extract impact usage
 for each pair of changes Δ, Δ′ //pair-wise comparison
   if Sim(IU(Δ), IU(Δ′)) ≥ σ3 //if impacts are similar
      RBF.add(Δ, Δ′) //report as recurring fixes
```

Figure 9: Recurring Fixes Recognition

Methodology IV

```
RecommendFix(X, ΔX)
 for each Y ∈ PeerOf(X) //for each peer of X
   X* = Affect(X, ΔX) //detect affected sub-trees of X
   M = Map(X*, Y)  //map them and other code elements to Y
   for each mapped pair (x, y) ∈ M //for the mapped elements
     O = DeriveOperation(x,y) //derive the relevant operation
     Recommend(O) //to recommend
```

Figure 10: Fixing Recommendation for Code Peers

Evaluation Results

Recurring Fix Recognition:

- Precision: 81%
- Recall: 74%

System	Class	Method	RBF	Prec.	Rec.	Fscore
ArgoUML	1063	2318	390	65%	70%	67%
Columba	1161	829	377	87%	73%	79%
ZK	295	490	188	91%	80%	85%
FlashRecruit	665	1007	244	84%	75%	79%
gEclipse	672	1126	215	78%	70%	74%

Table 4: Recognition Accuracy

Fixing Recommendation:

- Recall: 71%
- Precision: 49%

System	Check	Recom.	Correct	Prec.	Rec.	Fscore
ArgoUML	283	515	217	42%	77%	54%
Columba	199	293	139	47%	70%	56%
ZK	69	103	44	43%	64%	51%
FlashRecuit	65	77	39	51%	60%	55%
gEclipse	152	206	127	62%	84%	71%

Table 5: Recommendation Accuracy

Contributions

1. An Empirical study on Recurring Bug Fixes
2. New concepts, rules and algorithms to find, and fix recurring bugs
3. An Empirical Evaluation validating their approach

Limitations

1. Unreported Bug Fixes may remain in the dataset

2. Bugs may be Human biased when checking for what constitutes a bug

Conclusion

- Investigated Recurring bug-fixing changes
- Found a high amount of these bug fixes occur in code peers
- Developed novel algorithms to find and fix such recurring bug-fixes

Thank You