Key Concepts of this Paper

- GUI based application testing.
- Mapping with annotated EFG.
- Using reverse engineering with human support.

Motivation

- Large number of tests may become unusable each time the software is modified.
- Because of modified GUI restriction or the checkpoint validation.

Approach Overview:

- Ripping - Mimicking human user. This is needed for getting EFG.
- Mapping - It happened between GUI object and logical event.
- Repairing - Repair missing event & missing edge.
5. **Repairing**
- Repair missing event, when there is a NULL event in the sequence.
- Repair missing edge when no EFG edge is found for two consecutive events.
- If no path is found, then manually specify a path, adding events, edges, etc.

6. **Logical view of SITAR:**

7. **Algorithm Steps**
- Repair missing event, when there is a NULL event in the sequence, the tester needs to delete the event from the script. And a may-follow relationship will be added from the event prior to current event.
- The new event will be added to the vertex set and a pair of new edges related to this event will be added to the may-follow edge set.
- Now for this, if event's widget change like location or label, then the information will be added to a mapping table which can be referenced by future repairs.

8. **Algorithm Steps**
- Repair missing edge when no EFG edge is found for two consecutive events in the sequence. If there is a dominates edge in the annotated EFG then insertion happen in the sequence unless previous step occurs recursively.
- The repairer employs the shortest path algorithm to find a possible path. If no path is found, then the tester needs to manually specify a path, adding events, edges etc.
- The tester may create new entries for the approved paths table.
Algorithm

1. Initialization of global variables:
   \( G_0 = (V, E, R) \), mappingTable = ∅, approvedTable = ∅
2. Output: TS1, updated G1, mappingTable and approvedTable

Procedure repair(TestScript TS0):
3. TS1 ← ∅
4. For all test cases TC = \( \{s_1, s_2, \ldots, s_n\} \) ∈ TS0:
5.   enterSet ← ∅, exitSet ← ∅
6.   For all statements or checkpoints \( s_i \) ∈ TC:
7.     Map \( s_i \) to event \( e_i \) or checkpoint \( c_i \)
8.     enterSet.add(e_i), exitSet.add(c_i)
9.   For all events \( e_i \) and checkpoints \( c_i \) ∈ enterSet ∪ exitSet:
10.    If \( e_i = \text{NULL} \) or \( c_i = \text{NULL} \)
11.       repairSeg ← repairEventAndUpdateModel(e_i, e_{i-1}, e_{i+1})
12.       repairSeg.replace((e_i, e_{i-1}, e_{i+1}), repairedSeg)
13.   For all \( (e_i, e_{i-1}) \) where \( e_i \) and \( e_{i+1} \) ∈ exitSet:
14.      repairEdgeAndUpdateModel(e_i, e_{i-1}, e_{i+1})
15.      repairSeg = repairEventAndUpdateModel(e_i, e_{i-1}, e_{i+1})
16.      repairEdgeAndUpdateModel(e_i, e_{i-1}, e_{i+1})
17.   TS1.add(enterSet ∪ exitSet).mapToTestScript(y)
18.   Return TS1

Procedure repairEdgeAndUpdateModel(e_i, e_{i-1}, e_{i+1}):
20.   If confirm script \( e_i \) or remap to \( e_{i-1} \):
21.       add \( e_i \) or \( e_{i-1} \) to \( E_{\text{new}} \)
22.       add \( (e_i, e_{i-1}) \) and \( (e_{i-1}, e_{i+1}) \) to \( E_{\text{new}} \)
23.       If remap: mappingTable.add(s_i → e_i)
24.       Return \( e_i \) in \( E_{\text{new}} \)
25.   If deleteNode:
26.       add \( (e_i, e_{i-1}) \) to \( E_{\text{new}} \) if not exist
27.       return \( e_i \) in \( E_{\text{new}} \)
28.       return repairEdgeAndUpdateModel(e_{i-1}, e_{i+1})
29. Procedure repairEventAndUpdateModel(e_i, e_{i-1}, e_{i+1}):
30.   If approvedTable.lookup((v, y)) → \xi
31.      If \( (v, y) \in E_{\text{new}} \)
32.      return \( (v, y) \)
33.      return repairEventAndUpdateModel(e_i, e_{i-1}, e_{i+1})
34.      search the shortest path \((v, \xi, y) \in E\)
35.      tester confirm \((v, \xi, y) \) from suggested paths and may make manual modifications to the best suggested paths:
36.      approvedTable.add((v, y) → \xi)
37.      return \((v, \xi, y) \)

Overview of SITAR via PMS

Ripped EFG for version 2
Overview of SITAR via PMS (Test script)

Overview of SITAR via PMS (Regression)

Overview of SITAR via PMS (Mapping)

(EFG of version 2 after repairing TS2)
Annotated EFG of version 2 after TS3

Overview of SITAR via PMS (Repaired)

Experiment
- 370 QTP test scripts containing a total of 13,043 events and 1,224 checkpoints obtained by more than 200 testers on 3 software applications.

- Here used 2 versions of each application. The changes to the software made all test scripts unusable for 2 applications.

Experiment Result
- 89% of the scripts were repaired by SITAR.

- The resulting line coverage of the scripts for one application went up from 0% to 68.3%.
**Conclusion**

- SITAR successfully repairs test cases without breaking the business logic of the original test cases by accumulating human input as assertions in the model.

- SITAR achieves better automation over the lifetime of the overall repair process.

**Thank You**

Any Question Please?