1 Renaming Handler

1.1 Early Concepts

1.1.1 Possibly renamed without body changes nodes

$R_{wobc}(T, B) = \{b \in B \,|\, (\neg\exists t \in T\,(t.id = b.id))) \wedge (\exists t \in T\,(t.body = b.body))\}$

1.1.2 Possibly deleted or renamed with body changes nodes

$DR_{wbc}(T, B) = \{b \in B \,|\, \neg\exists t \in T\,(t.id = b.id \vee t.body = b.body)\}$

1.2 Match Algorithm

Algorithm 1: Match Algorithm

Input: L, B, R, M
Output: Set of quadruples (l, b, r, m) consisting of the base node b and its corresponding left node l, right node r and merge node m

```
1  matches ← ∅;
2  foreach b in possiblyRenamedOrDeletedBaseNodes(L, B, R) do
3  |  l ← getCorrespondentNode(b, L);
4  |  r ← getCorrespondentNode(b, R);
5  |  m ← getMergeNode(l, r, M);
6  |  matches ← matches ∪ (l, b, r, m);
7  end
8  return matches
9  function possiblyRenamedOrDeletedBaseNodes(L, B, R)
10 |  return DR_wbc(L, B) ∪ DR_wbc(R, B) ∪ R_wobc(L, B) ∪ R_wobc(R, B);
11 end
12 function getCorrespondentNode(b, T)
13 |  t ← findFirst(t ∈ T → t.id = b.id);
14 |  if t = null then
15 |  |  t ← findFirst(t ∈ T → t.body = b.body);
16 |  end
17 |  if t = null then
18 |  |  t ← findFirst(t ∈ T → t.body ≈ b.body ∧ ( t.id.name = b.id.name∨
   |  |    t.id.params = b.id.params));
19 |  end
20 |  if t = null then
21 |  |  t ← findFirst(t ∈ T → t.body = substring(b.body) ∨ b.body = substring(t.body));
22 |  end
23 |  return t;
24 end
25 function getMergeNode(l, r, M)
26 |  if l ≠ null then
27 |  |  return find(m ∈ M → m.id = l.id);
28 |  end
29 |  if r ≠ null then
30 |  |  return find(m ∈ M → m.id = r.id);
31 |  end
32 |  return null;
33 end
```

1.3 Handle Algorithms

Algorithm 2: Check References and Merge Methods Variant

Input: (l, b, r, m), M

```
1  if singleRenamingOrDeletion(l, b, r) then
2  |  m.body = textualMerge(l, b, r);
3  |  removeUnmatchedNode(l, r, m, M);
4  else if l.id ≠ r.id then
5  |  m.body = conflit(l, b, r);
6  |  removeUnmatchedNode(l, r, m, M);
7  else if l.body ≠ r.body then
8  |  if newReferenceTo(l) ∨ newReferenceTo(r) then
9  |  |  m.body = conflict(l, b, r);
10 |  |  removeUnmatchedNode(l, r, m, M);
11 |  else
12 |  |  m.body = textualMerge(l, b, r);
13 |  |  removeUnmatchedNode(l, r, m, M);
14 |  end
15 end
16 function singleRenamingOrDeletion(l, b, r)
17 |  return l.id = b.id ∨ r.id = b.id;
18 end
19 function removeUnmatchedNode(l, r, m, M)
20 |  if l.id = m.id ∧ r.id ≠ m.id then
21 |  |  removeNode(r, M);
22 |  end
23 end
```

Algorithm 3: Merge Methods Variant

Input: (l, b, r, m), M

```
1  m.body = textualMerge(l, b, r);
2  removeUnmatchedNode(l, r, m, M);
3  function removeUnmatchedNode(l, r, m, M)
4  |  if l.id = m.id ∧ r.id ≠ m.id then
5  |  |  removeNode(r, M);
6  |  end
7  end
```

Algorithm 4: Check Textual and Keep Both Methods Variant

Input: (l, b, r, m), M

```
1  if singleRenamingOrDeletion(l, b, r) then
2  |  if textualMergeHasConflictInvolvingSignature(b) then
3  |  |  m.body = conflict(l, b, r);
4  |  |  removeUnmatchedNode(l, r, m, M);
5  |  end
6  else if l.id ≠ r.id ∧ l.body = r.body then
7  |  m.body = conflict(l, b, r);
8  |  removeUnmatchedNode(l, r, m, M);
9  end
10 function singleRenamingOrDeletion(l, b, r)
11 |  return l.id = b.id ∨ r.id = b.id;
12 end
13 function removeUnmatchedNode(l, r, m, M)
14 |  if l.id = m.id ∧ r.id ≠ m.id then
15 |  |  removeNode(r, M);
16 |  end
17 end
```

Algorithm 5: Keep Both Methods Variant

Input: (l, b, r, m), M

```
1  if singleRenamingOrDeletion(l, b, r) ∧ hasConflict(m) then
2  |  removeConflict(m);
3  end
4  function singleRenamingOrDeletion(l, b, r)
5  |  return l.id = b.id ∨ r.id = b.id;
6  end
```