1 Renaming Handler

1.1 Early Concepts

1.1.1 Possibly renamed without body changes nodes

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

1.1.2 Possibly deleted or renamed with body changes nodes

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

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  $matches \leftarrow \emptyset$;
2  foreach $b$ in possiblyRenamedOrDeletedBaseNodes($L$, $B$, $R$) do
3      $l \leftarrow$ getCorrespondentNode($b$, $L$);
4      $r \leftarrow$ getCorrespondentNode($b$, $R$);
5      $m \leftarrow$ getMergeNode($l$, $r$, $M$);
6      $matches \leftarrow matches \cup (l, b, r, m)$;
7  end
8  return $matches$
9  function possiblyRenamedOrDeletedBaseNodes($L$, $B$, $R$)
10     return $DR_{wbc}(L, B) \cup DR_{wbc}(R, B) \cup R_{wobc}(L, B) \cup R_{wobc}(R, B)$;
11 function getCorrespondentNode($b$, $T$)
12     $t \leftarrow findFirst(t \in T \rightarrow t.id = b.id)$;
13     if $t = null$ then
14         $t \leftarrow findFirst(t \in T \rightarrow t.body = b.body)$;
15     end
16     if $t = null$ then
17         $t \leftarrow findFirst(t \in T \rightarrow t.body \approx b.body \wedge ($ $t.id.name = b.id.name \vee$
             $t.id.params = b.id.params))$;
18     end
19     if $t = null$ then
20         $t \leftarrow findFirst(t \in T \rightarrow t.body = substring(b.body) \vee b.body = substring(t.body))$;
21     end
22     return $t$;
23 function getMergeNode($l$, $r$, $M$)
24     if $l \neq null$ then
25         return $find(m \in M \rightarrow m.id = l.id)$;
26     end
27     if $r \neq null$ then
28         return $find(m \in M \rightarrow m.id = r.id)$;
29     end
30     return $null$;
```

1.3 Handle Algorithms

Algorithm 2: Check References and Merge Methods Variant

```
Input: (l, b, r, m), M
if singleRenamingOrDeletion($l$, $b$, $r$) then
    $m.body = textualMerge(l, b, r)$;
    removeUnmatchedNode($l$, $r$, $m$, $M$);
else if $l.id \neq r.id$ then
    $m.body = conflit(l, b, r)$;
    removeUnmatchedNode($l$, $r$, $m$, $M$);
else if $l.body \neq r.body$ then
    if newReferenceTo($l$) $\vee$ newReferenceTo($r$) then
        $m.body = conflict(l, b, r)$;
        removeUnmatchedNode($l$, $r$, $m$, $M$);
    else
        $m.body = textualMerge(l, b, r)$;
        removeUnmatchedNode($l$, $r$, $m$, $M$);
    end
end
function singleRenamingOrDeletion($l$, $b$, $r$)
    return $l.id = b.id \vee r.id = b.id$;
function removeUnmatchedNode($l$, $r$, $m$, $M$)
    if $l.id = m.id \wedge r.id \neq m.id$ then
        $removeNode(r, M)$;
    end
```

Algorithm 3: Merge Methods Variant

```
Input: (l, b, r, m), M
$m.body = textualMerge(l, b, r)$;
removeUnmatchedNode($l$, $r$, $m$, $M$);
function removeUnmatchedNode($l$, $r$, $m$, $M$)
    if $l.id = m.id \wedge r.id \neq m.id$ then
        $removeNode(r, M)$;
    end
```

Algorithm 4: Check Textual and Keep Both Methods Variant

```
Input: (l, b, r, m), M
if singleRenamingOrDeletion($l$, $b$, $r$) then
    if textualMergeHasConflictInvolvingSignature($b$) then
        $m.body = conflict(l, b, r)$;
        removeUnmatchedNode($l$, $r$, $m$, $M$);
    end
else if $l.id \neq r.id \wedge l.body = r.body$ then
    $m.body = conflict(l, b, r)$;
    removeUnmatchedNode($l$, $r$, $m$, $M$);
end
function singleRenamingOrDeletion($l$, $b$, $r$)
    return $l.id = b.id \vee r.id = b.id$;
function removeUnmatchedNode($l$, $r$, $m$, $M$)
    if $l.id = m.id \wedge r.id \neq m.id$ then
        $removeNode(r, M)$;
    end
```

Algorithm 5: Keep Both Methods Variant

```
Input: (l, b, r, m), M
if singleRenamingOrDeletion($l$, $b$, $r$) $\wedge$ hasConflict($m$) then
    $removeConflict(m)$;
end
function singleRenamingOrDeletion($l$, $b$, $r$)
    return $l.id = b.id \vee r.id = b.id$;
```