Unit Test Generation Using Tree Mining Algorithms

Julien Lienard

Kim Mens

Siegfried Nijssen

table of contents

Introduction

- Identify bad practices and common mistakes in code
- Improve code quality
- Huge code data bases

Baseline Paper

- Mine patterns in CS1 student code
- Use Tree mining algorithm
- Small evaluation
- We want to go further

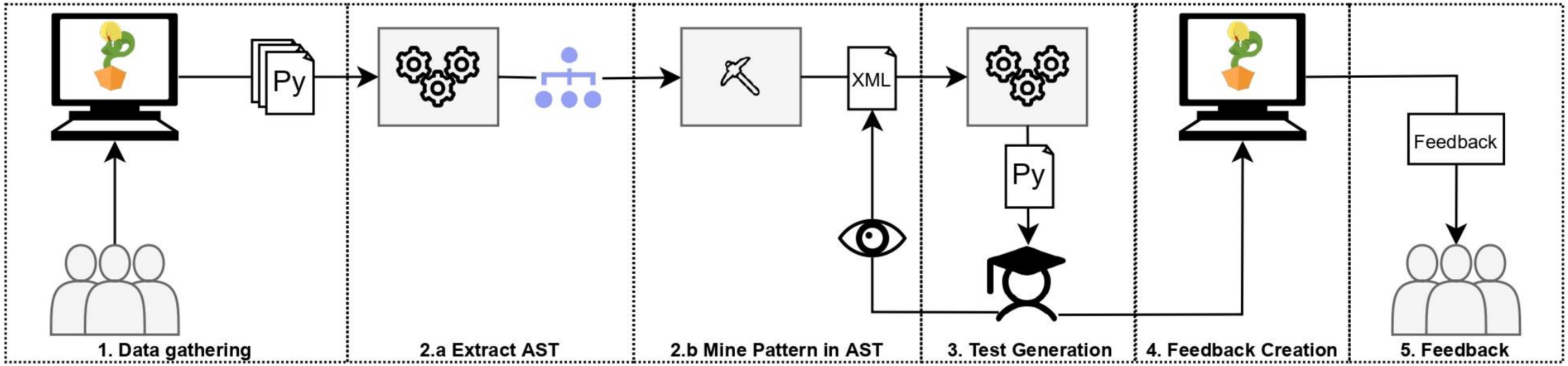
The Good, the Bad, and the Ugly: Mining for Patterns in Student Source Code

K. Mens, S. Nijssen, H.S. Pham

What do we add

- More systematic analysis but still manual
- Use of observed solutions for future students
- Automation of the detection of patterns of interest
- Creation of feedbacks to improve code quality

Methodology

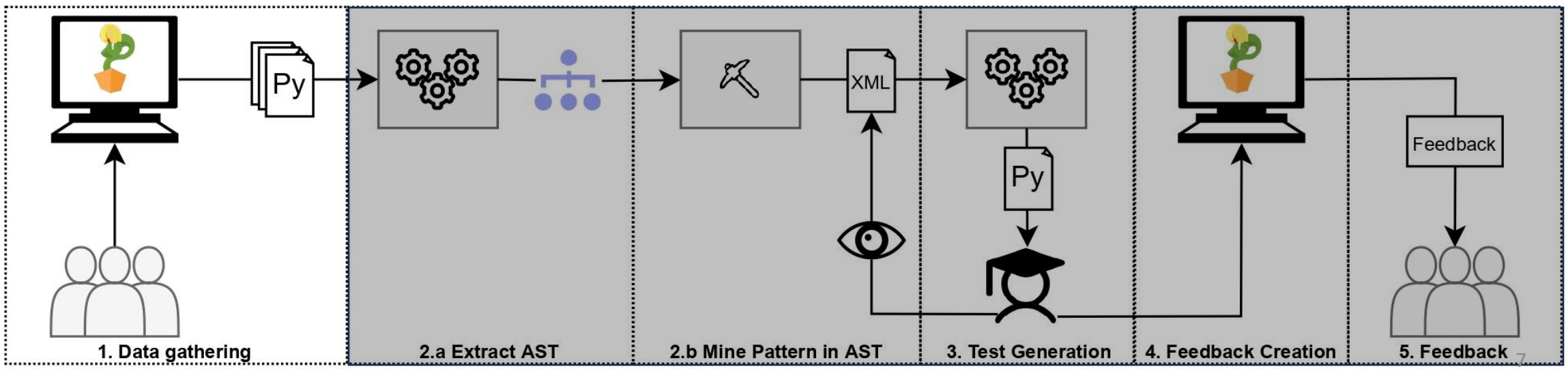


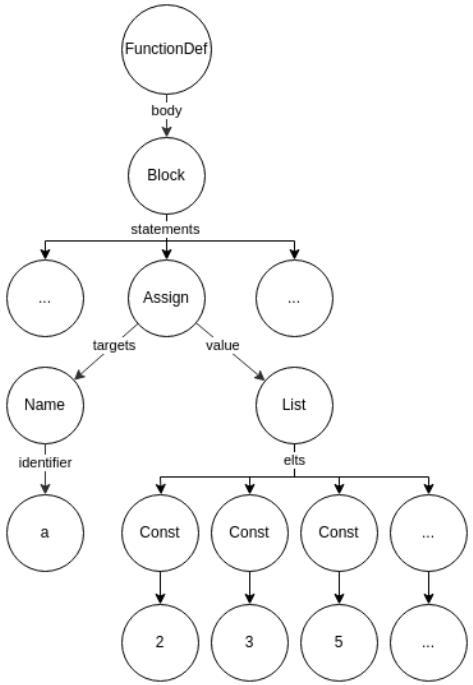
```python
def facteurs(n):
    assert isinstance(n,int),"please enter an integer"
    assert (n>0),"please enter a positive number"
    a=[2,3,5,7,11,13]
    count=0
    for i in a:
        while n%i==0:
            count+=1
            n=n//i
    if count==0 and n!=1:
        count=1
    return (count)
```

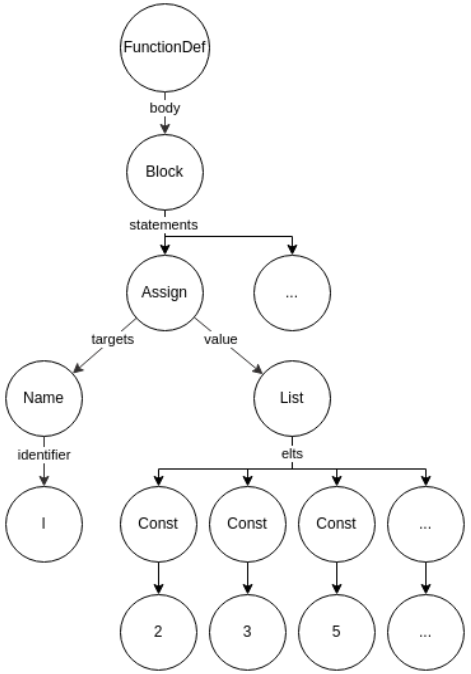
```python
def factors(n):
    l = [2,3,5,7,11,13,17,19] #list of primes
    i = 0
    t=[]
    for i in range(len(l)) :
        if n%l[i] == 0:
            t.append(l[i])
            i -= 1
    nbr = len(t)
    return (nbr)
```

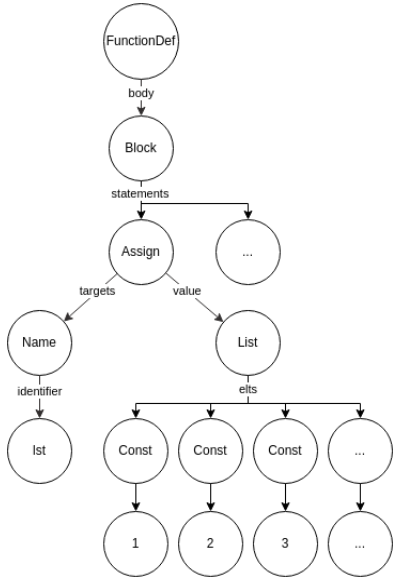
```python
def facteurs(n):
    lst=[1,2,3,5,7,11,13,19]

    for i in range(1,8):
        for l in range(0,8):
            for k in range(0,8):
                for j in range(0,8):
                    if lst[i]*lst[j]*lst[k]*lst[l]==n:
                        if l==0:
                            return 2
                        elif k==0:
                            return 3
                        elif j==0:
                            return 4
                        else:
                            return 5
```

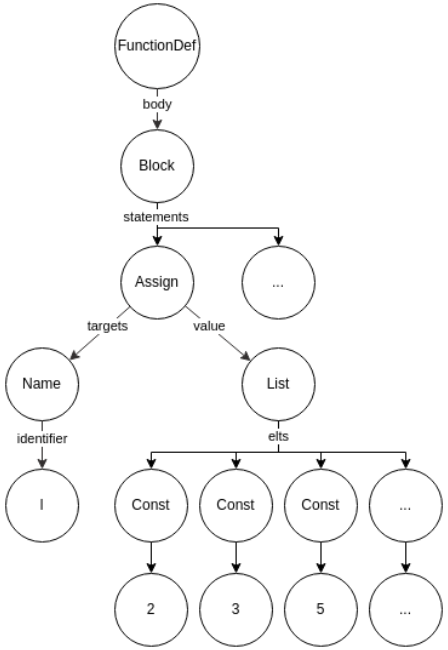
```python
def facteurs(n):
    l = [2,3,5,7,11,13,17,23,29,31]
    f = 0
    for i in range (2,n//2):
        f = 0
        if n == 1:
            None
        elif n%i == 0:
                for j in l:
                    while n%j == 0:
                        f += 1
                        n = n/j
                break
        else:
            f = 1
    return f
```

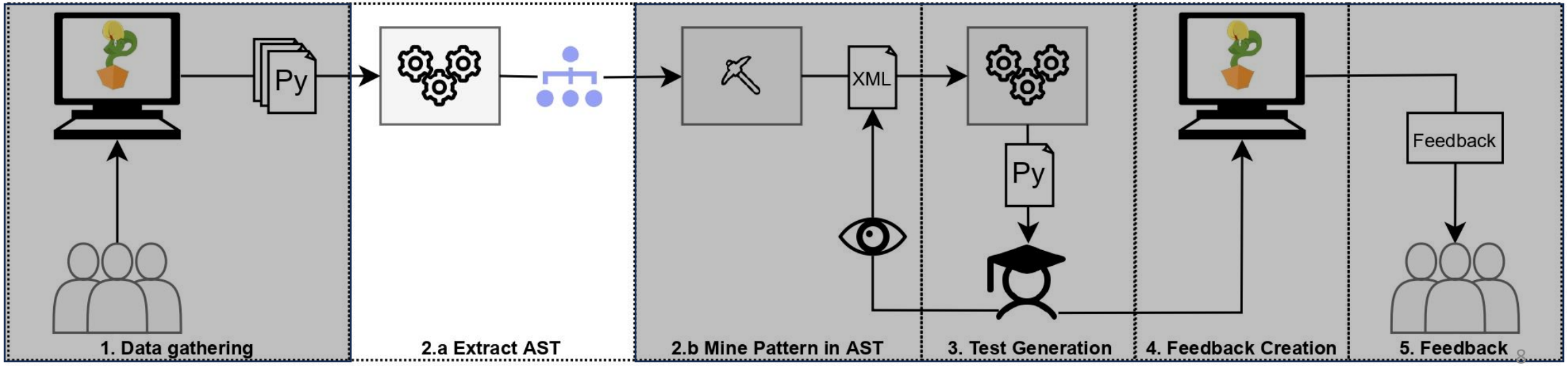


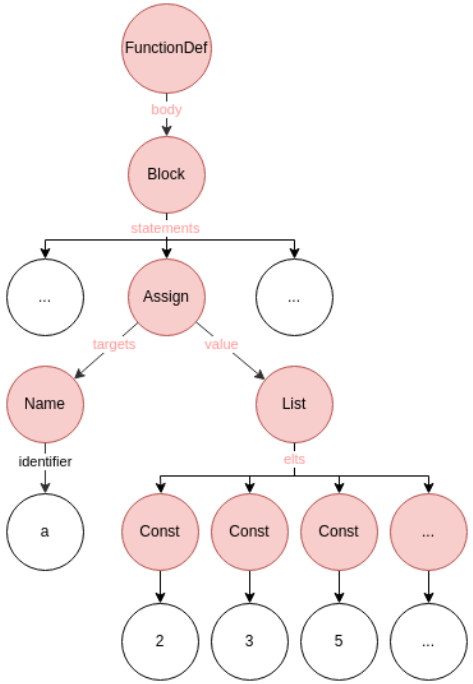

FunctionDef
body
Block
statements
...
Assign
...
targets
value
Name
List
identifier
elts
a
Const
Const
Const
...
2
3
5
...

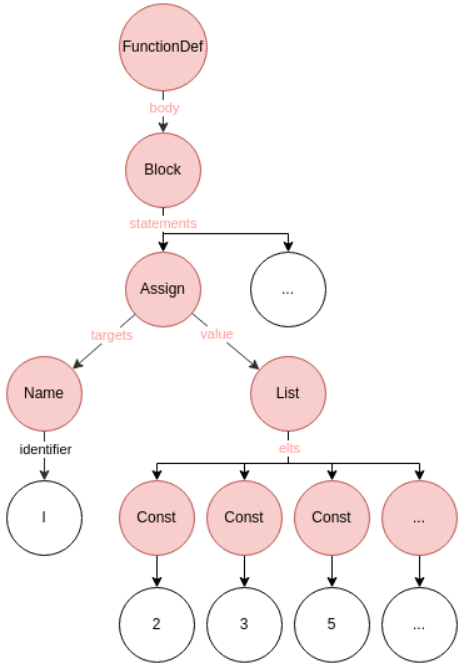

FunctionDef
body
Block
statements
Assign
...
targets
value
Name
List
identifier
elts
I
Const
Const
Const
...
2
3
5
...

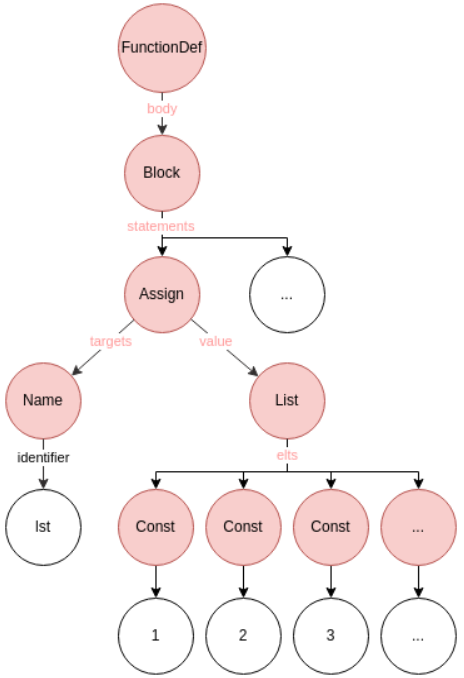

FunctionDef
body
Block
statements
Assign
...
targets
value
Name
List
identifier
elts
lst
Const
Const
Const
...
1
2
3
...

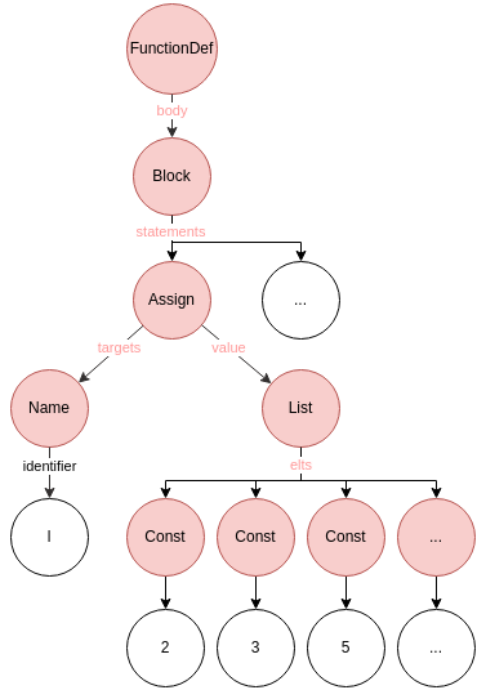

FunctionDef
body
Block
statements
Assign
...
targets
value
Name
List
identifier
elts
I
Const
Const
Const
...
2
3
5
...

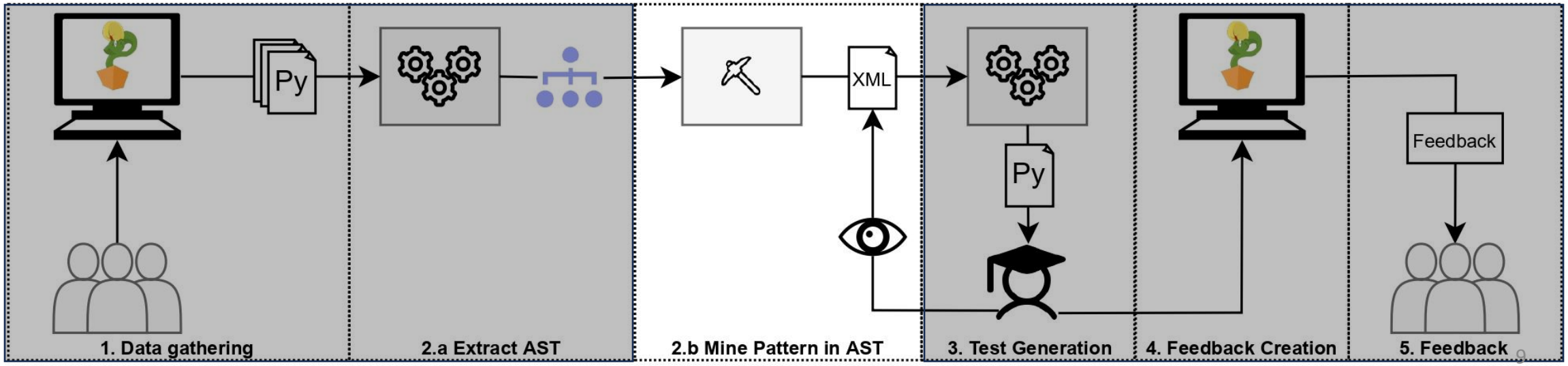

Py
XML
Py
Feedback
1. Data gathering
2.a Extract AST
2.b Mine Pattern in AST
3. Test Generation
4. Feedback Creation
5. Feedback


FunctionDef
body
Block
statements
...
Assign
...
targets
value
Name
List
identifier
elts
a
Const
Const
Const
...
2
3
5
...

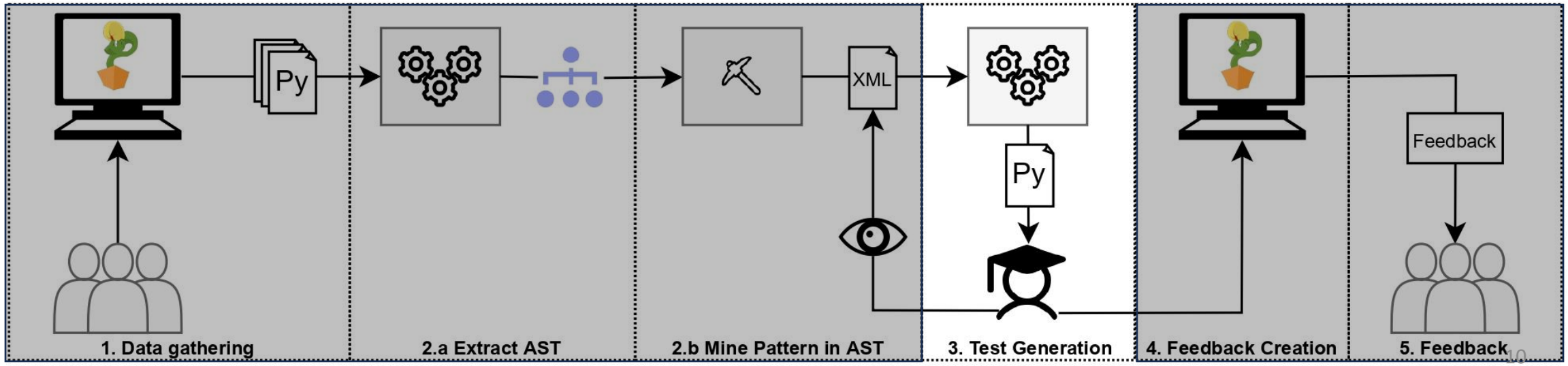

FunctionDef
body
Block
statements
Assign
...
targets
value
Name
List
identifier
elts
l
Const
Const
Const
...
2
3
5
...


FunctionDef
body
Block
statements
Assign
...
targets
value
Name
List
identifier
elts
lst
Const
Const
Const
...
1
2
3
...

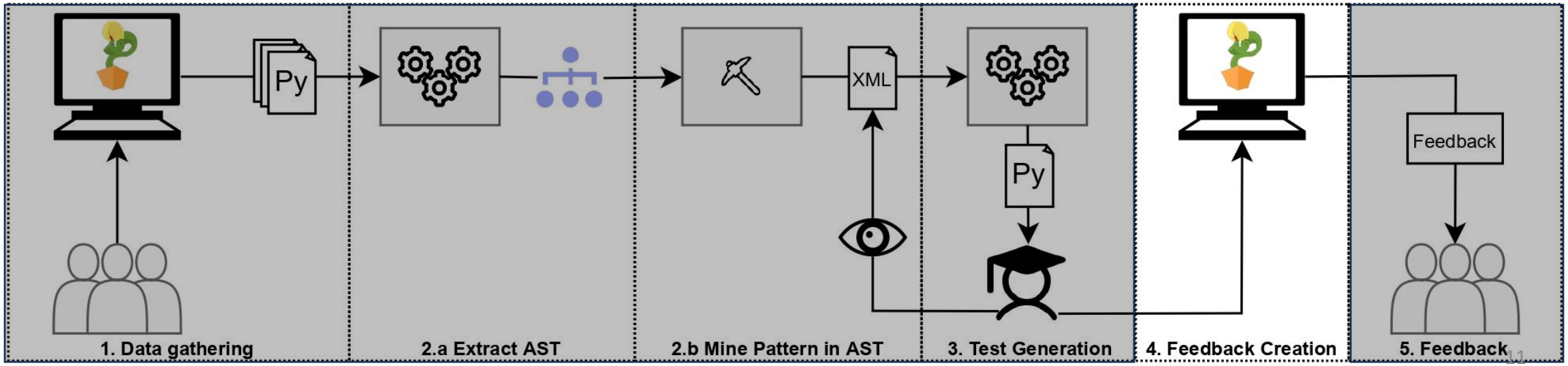

FunctionDef
body
Block
statements
Assign
...
targets
value
Name
List
identifier
elts
l
Const
Const
Const
...
2
3
5
...


Py
XML
Py
Feedback
1. Data gathering
2.a Extract AST
2.b Mine Pattern in AST
3. Test Generation
4. Feedback Creation
5. Feedback

```python
def match_pattern_2(myast):
    ...
    for val in ast.iter_child_nodes(assign1) if isinstance(val, ast.Name)]
      if len(names1) < 1:
          continue
      lists1 = [val for val in ast.iter_child_nodes(assign1) if isinstance(val, ast.List)]
      if len(lists1) < 1:
          continue
      for list1 in lists1:
          constants1 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants1) < 1:
              continue
          constants2 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants2) < 2:
              continue
          constants3 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants3) < 3:
              continue
          constants4 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants4) < 4:
              continue
          constants5 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants5) < 5:
              continue
          constants6 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants6) < 6:
              continue
          return True
    return False
```

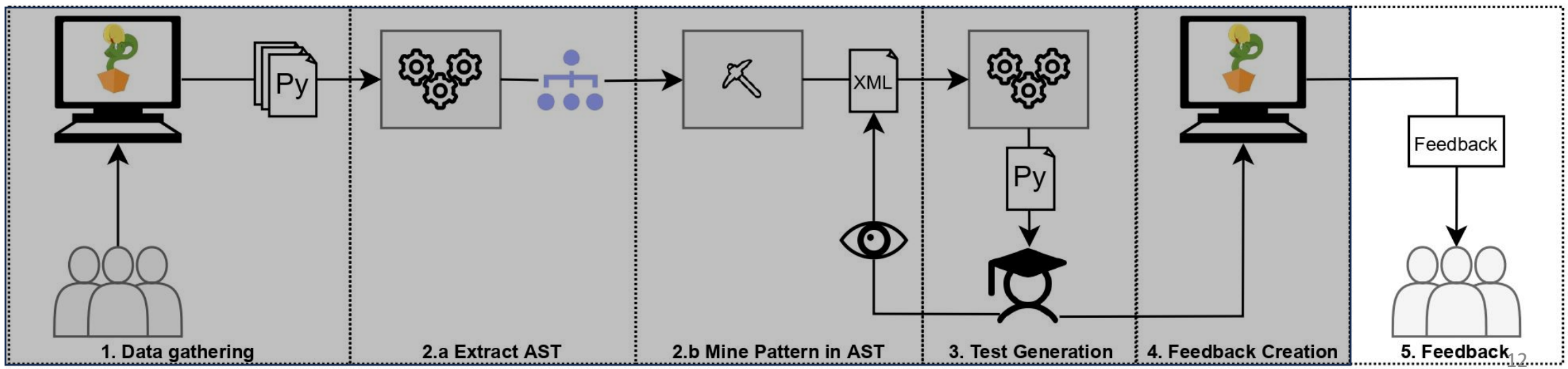


```
class TestQ(unittest.TestCase):
  def __init__(self,*args,**kwargs):
    super().__init__(*args, **kwargs)
    with open(str(q.__name__)+'.py','r') as f:
      self.ast = ast.parse(f.read())

  def test_hard_coded_list(self):
    if match_pattern_2(self.ast):
      self.fail("Feedback: Avoid using hard coded lists of precalculated numbers. Try to generalise your solution.")
```



There are some errors in your answer. Your score is 84.0%. [Submission #646e0c6638185572f5c04455]

```
Suggestion:
Avoid using hard coded lists of precalculated numbers. Try to generalise your solution.

----------------------------------------------------------------------
Ran 9 tests in 0.007s

FAILED (failures=4)Failed test:
2 != 3 : Wrong value for factors(2022)

Failed test:
2 != 3 : Wrong value for factors(2022)

Failed test:
0 != 1 : Wrong value for factors(5)
```



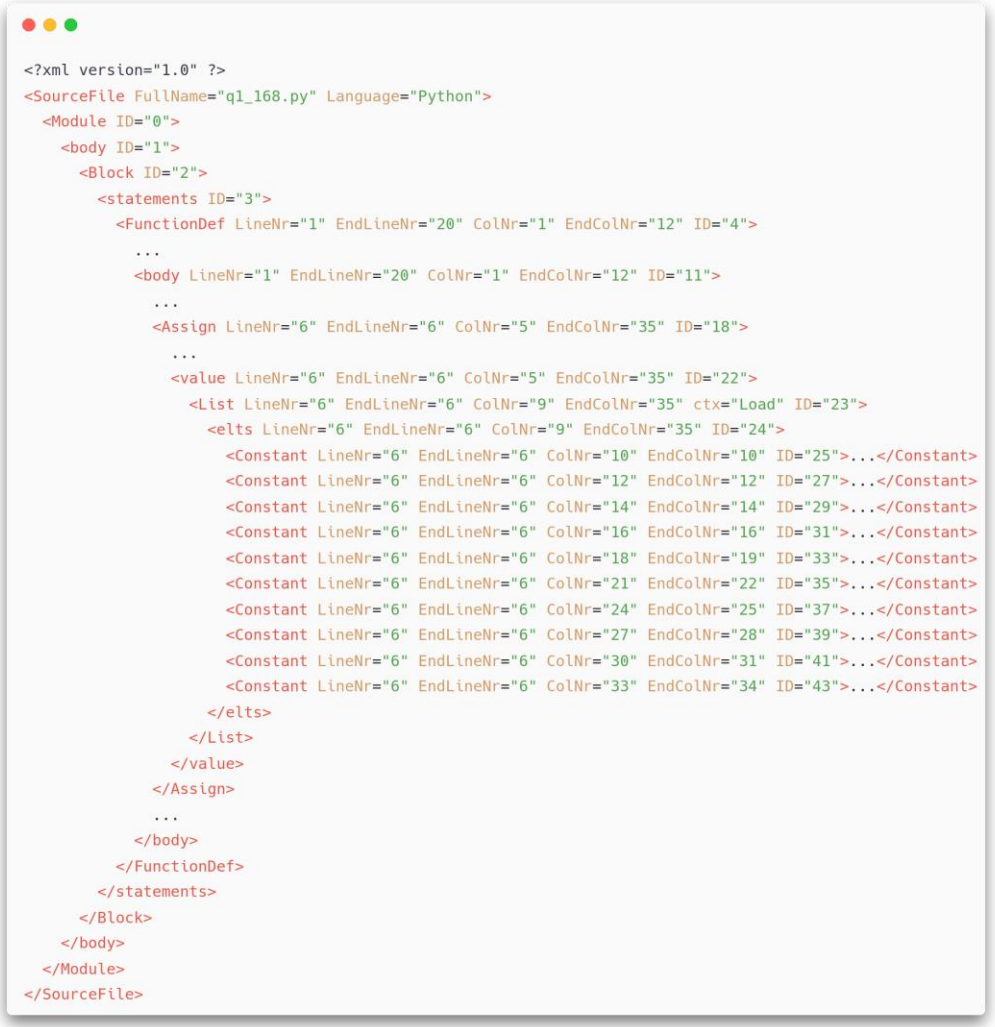
Step 1: Data gathering

- Student code submissions
- Exam question
- No feedback during the exam, ensuring unbiased patterns
- Similar exam conditions and individual responses of students
- Two groups: score ≥50% and <50%

```
def factors(n):
    l = [2,3,5,7,11,13,17,19] #list of primes
    i = 0
    t=[]
    for i in range(len(l)) :
        if n%l[i] == 0:
            t.append(l[i])
            i -= 1
    nbr = len(t)
    return (nbr)
```

Step 2a: Extract AST

- Work on AST
- Tree-like structure
- Generate XML files representing the AST
- Applicable to other languages

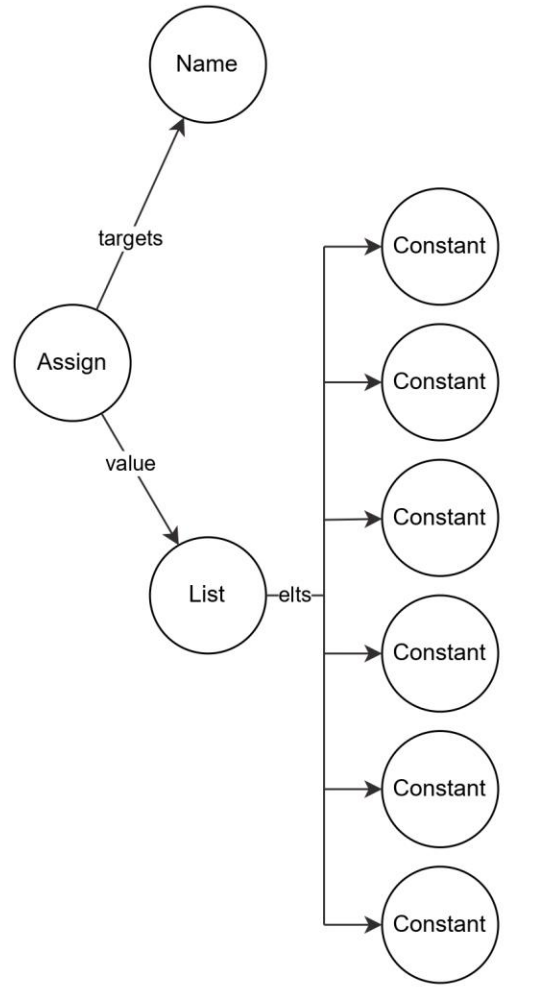


```
<?xml version="1.0" ?>
<SourceFile FullName="q1_168.py" Language="Python">
  <Module ID="0">
    <body ID="1">
      <Block ID="2">
        <statements ID="3">
          <FunctionDef LineNr="1" EndLineNr="20" ColNr="1" EndColNr="12" ID="4">
            ...
            <body LineNr="1" EndLineNr="20" ColNr="1" EndColNr="12" ID="11">
              ...
              <Assign LineNr="6" EndLineNr="6" ColNr="5" EndColNr="35" ID="18">
                ...
                <value LineNr="6" EndLineNr="6" ColNr="5" EndColNr="35" ID="22">
                  <List LineNr="6" EndLineNr="6" ColNr="9" EndColNr="35" ctx="Load" ID="23">
                    <elts LineNr="6" EndLineNr="6" ColNr="9" EndColNr="35" ID="24">
                      <Constant LineNr="6" EndLineNr="6" ColNr="10" EndColNr="10" ID="25">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="12" EndColNr="12" ID="27">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="14" EndColNr="14" ID="29">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="16" EndColNr="16" ID="31">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="18" EndColNr="19" ID="33">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="21" EndColNr="22" ID="35">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="24" EndColNr="25" ID="37">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="27" EndColNr="28" ID="39">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="30" EndColNr="31" ID="41">...</Constant>
                      <Constant LineNr="6" EndLineNr="6" ColNr="33" EndColNr="34" ID="43">...</Constant>
                    </elts>
                  </List>
                </value>
              </Assign>
              ...
            </body>
          </FunctionDef>
        </statements>
      </Block>
    </body>
  </Module>
</SourceFile>
```

Step 2b: Mine pattern in AST

- FREQTALS is a tree mining algorithm for identifying frequent patterns in program ASTs.
- Configured with constraints like pattern size
- Searches for patterns based on structural characteristics and label frequencies.
- Discovers control flow, data structure, and algorithm patterns.
- Output patterns as tree-like structure

```
<Assign>
  <__directives>
    <match-sequence/>
  </__directives>
  <targets>
    <Name>
      <identifier>
        <Dummy>
          ...
        </Dummy>
      </identifier>
    </Name>
  </targets>
  <value>
    <List>
      <elts>
        <__directives>
          <match-sequence/>
        </__directives>
        <Constant>
          <identifier>
            <Dummy>
              ...
            </Dummy>
          </identifier>
        </Constant>
        ...
      </elts>
    </List>
  </value>
</Assign>
```



Step 3: Test generation

- Generate python function from the tree representation of the pattern
- Take an AST of student's code as input
- Return True if the AST matches the pattern
- Match a specific pattern
- Ensures equivalence with patterns

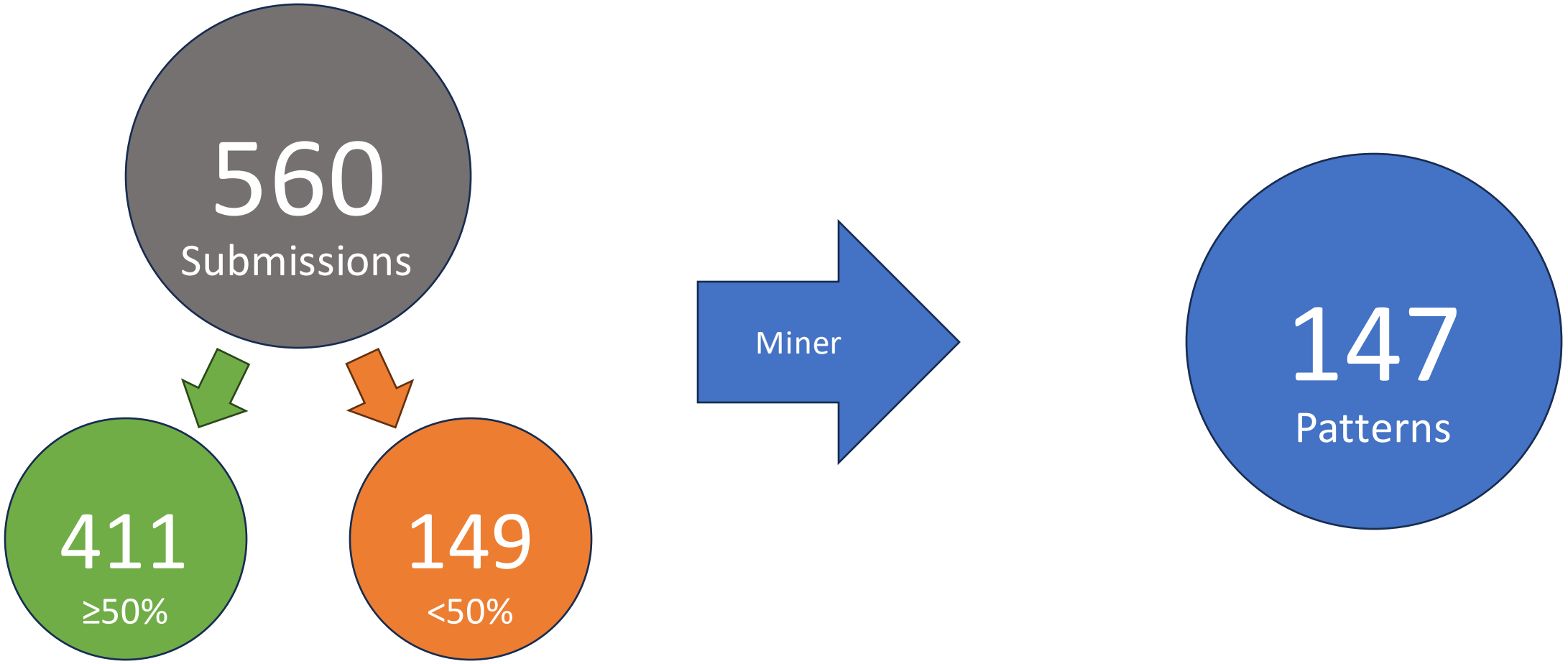
```
def match_pattern_2(myast):
    ...
    for val in ast.iter_child_nodes(assign1) if isinstance(val, ast.Name)]
      if len(names1) < 1:
          continue
      lists1 = [val for val in ast.iter_child_nodes(assign1) if isinstance(val, ast.List)]
      if len(lists1) < 1:
          continue
      for list1 in lists1:
          constants1 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants1) < 1:
              continue
          constants2 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants2) < 2:
              continue
          constants3 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants3) < 3:
              continue
          constants4 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants4) < 4:
              continue
          constants5 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants5) < 5:
              continue
          constants6 = [val for val in ast.iter_child_nodes(list1) if isinstance(val, ast.Constant)]
          if len(constants6) < 6:
              continue
          return True
    return False
```

Step 4: Feedback creation

- Improve feedbacks in auto graders
- Create simple unit test calling the generated function
- Include in our auto grader
- Easy to add test for other patterns

```
class TestQ(unittest.TestCase):
  def __init__(self,*args,**kwargs):
    super().__init__(*args, **kwargs)
    with open(str(q.__name__)+'.py','r') as f:
      self.ast = ast.parse(f.read())

  def test_hard_coded_list(self):
    if match_pattern_2(self.ast):
      self.fail("Feedback: Avoid using hard coded lists of precalculated numbers. Try to generalise your solution.")
```

Results



Good solution

- Compute prime factors
- Example of a good solution
- 3 main parts
  - Var initiation
  - **Loop** until n
  - **While** we can divide:
    - we divide our value
    - We **add 1** to our counter

```python
def facteurs(n):
    """
    Pre:  n est un entier strictement positif
    Post: retourne un entier représentant le nombre de facteurs premiers de n
    """
    nfact = 0
    val = n
    for i in range(2, n+1):
        while val % i == 0:
            val /= i
            nfact += 1
    return nfact
```

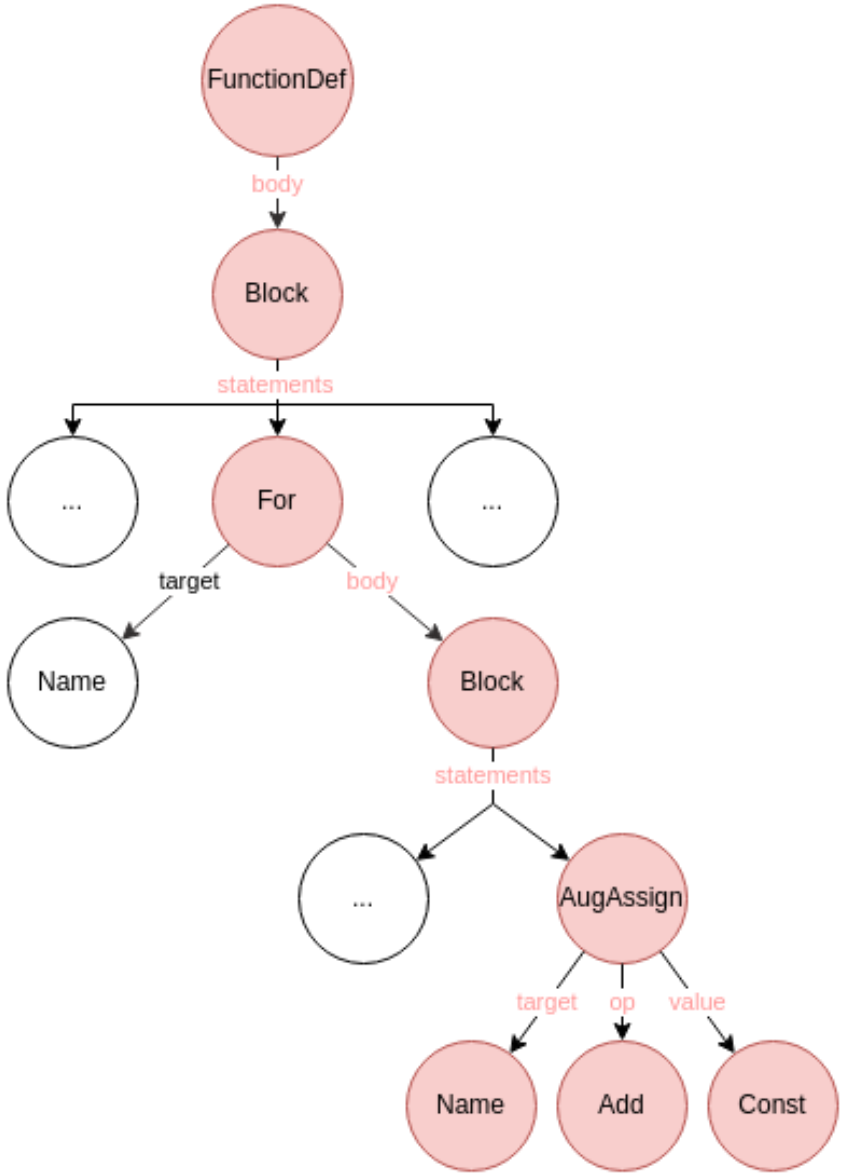
Examples of a mistake

```
def factors(n):
    for i in range(2,n):
        if n%i!=0:
            i+=1
            return 1
        elif n==1:
            return 0
    count = 0
    for f in range(2,n):
        while n%f==0:
            count+=1
            n=n/f
            return n
        f+=1
    return count
```

```
def factors(n):
    fac = 0
    if n == 1:
        fac = 0
    for i in range(2,n+1):
        sum = n/i
        sum2 = sum/i
        fac +=1
    return fac
```

```
def factors(n):
    count = 0
    if n == 1:
        return 0
    for i in range (2,n+1):
        if n % i == 0:
            n = n//i
        count += 1
    return count
```

Pattern found



Python test

```python
def match_pattern_79(myast):
    ...
    fors1 = [val for val in ast.iter_child_nodes(block1) if isinstance(val, ast.For)]
    if len(fors1) < 1:
        continue
    for for1 in fors1:
        ...
        augassigns1 = [val for val in ast.iter_child_nodes(for1) if isinstance(val, ast.AugAssign)]
        if len(augassigns1) < 1:
            continue
        for augassign1 in augassigns1:
            names3 = [val for val in ast.iter_child_nodes(augassign1) if isinstance(val, ast.Name)]
            if len(names3) < 1:
                continue
            adds1 = [val for val in ast.iter_child_nodes(augassign1) if isinstance(val, ast.Add)]
            if len(adds1) < 1:
                continue
            constants2 = [val for val in ast.iter_child_nodes(augassign1) if isinstance(val, ast.Constant)]
            if len(constants2) < 1:
                continue
            return True
    return False
```

Unit Test with feedback

```
class TestQ(unittest.TestCase):

  def __init__(self,*args,**kwargs):
    super().__init__(*args, **kwargs)
    with open(str(q.__name__)+'.py','r') as f:
      self.ast = ast.parse(f.read())

  def test_incr_in_for(self):
    if match_pattern_79(self.ast):
      self.fail("Feedback: Incrementing the factor counter inside your For loop might give you too high
numbers. Maybe you should update your counter only in certain cases?")
```

Conclusion

- Bad practices and common mistakes can be found in students' code
- Manual identification of interesting patterns is hard and time consuming
- Possibilities for advanced feedback creation
- Test are too technical to modify

Future work

- Further validation with more data, students, and teachers.
- The goal is to study how teachers use the tool and improve it based on their needs.
- Validation is also aimed at determining if students benefit from the improved feedback provided by the generated test suites.
- Automation of bad pattern detection
- Make it easier to modify unit tests

What's next?

- Language to describe pattern
- Build over Python
- Easy to learn and use
- Inspired by regex
- Backtracking approach
- ANTLR for Python with added grammar

```
def ?(foo):
  ?*
  ? = [?, ?, ?, ?, ?, ?, ?*]
  ?*
  return ?
```

Pattern language

Wildcard	Description
?	Match exactly 1 element
?*	Match 0 or more elements
?{name}	Match 1 element and bind it to {name}
?:	Match 1 element with a body
?*:	Match 0 or more element with bodies

Some examples

```
def ?(bar):
    ?:
        for ? in ?:
            ?*
    return ?
```

```
def ?(?*):
    ?*
    ?var1 = ?
    ?*
    return ?var1
```

```
def foo(bar):
    ?*
    ?*:
      ?*
      ?var1+=1
      ?*
    ?*
    return ?var1
```

```
def ?(foo):
  ?*
  ? = [?, ?, ?, ?, ?, ?, ?*]
  ?*
  return ?
```

Questions?

Bonus

FREQTALS

- Tree mining algorithm for frequent pattern identification.
- Uses XML tree format.
- Generates and prunes patterns iteratively.
- Configurable with minimum pattern size, etc.
- Allows to find specific syntactic constructs.

PHAM, Hoang Son, NIJSSEN, Siegfried, MENS, Kim, *et al.* Mining patterns in source code using tree mining algorithms.