



Hypergolic

DELIBERATE MACHINE LEARNING

*“If it looks good,
it flies good.”*

- Classic aviation quote



Code Smells in Data Science: What can we do about them?

- Why do we care?
- What is a Code Smell?
- How do we improve readability?
- **Relevant concepts**
 - Primitive Obsession
 - Dependency Injection
 - Guard Clauses
 - “The Happy Path”

About Me



- **past: finance, mobile gaming, head of DS**
Industrial scale NLP for investment banks
- **now: I teach DSes to write better code**
I am a startup ML consultant
- **blog: laszlo.substack.com**
- **community: cq4ds.com**

Why do we care?

- Programming is communication
- Communication needs a language
- We read more than we write
- Issues need standardised solutions
- Drive for productivity

What do we mean by “Code Smell”?

✗ Not a bug

✗ Doesn't need immediate attention

✗ Tech Debt vs Code Rot

✓ *_Might_* cause problem

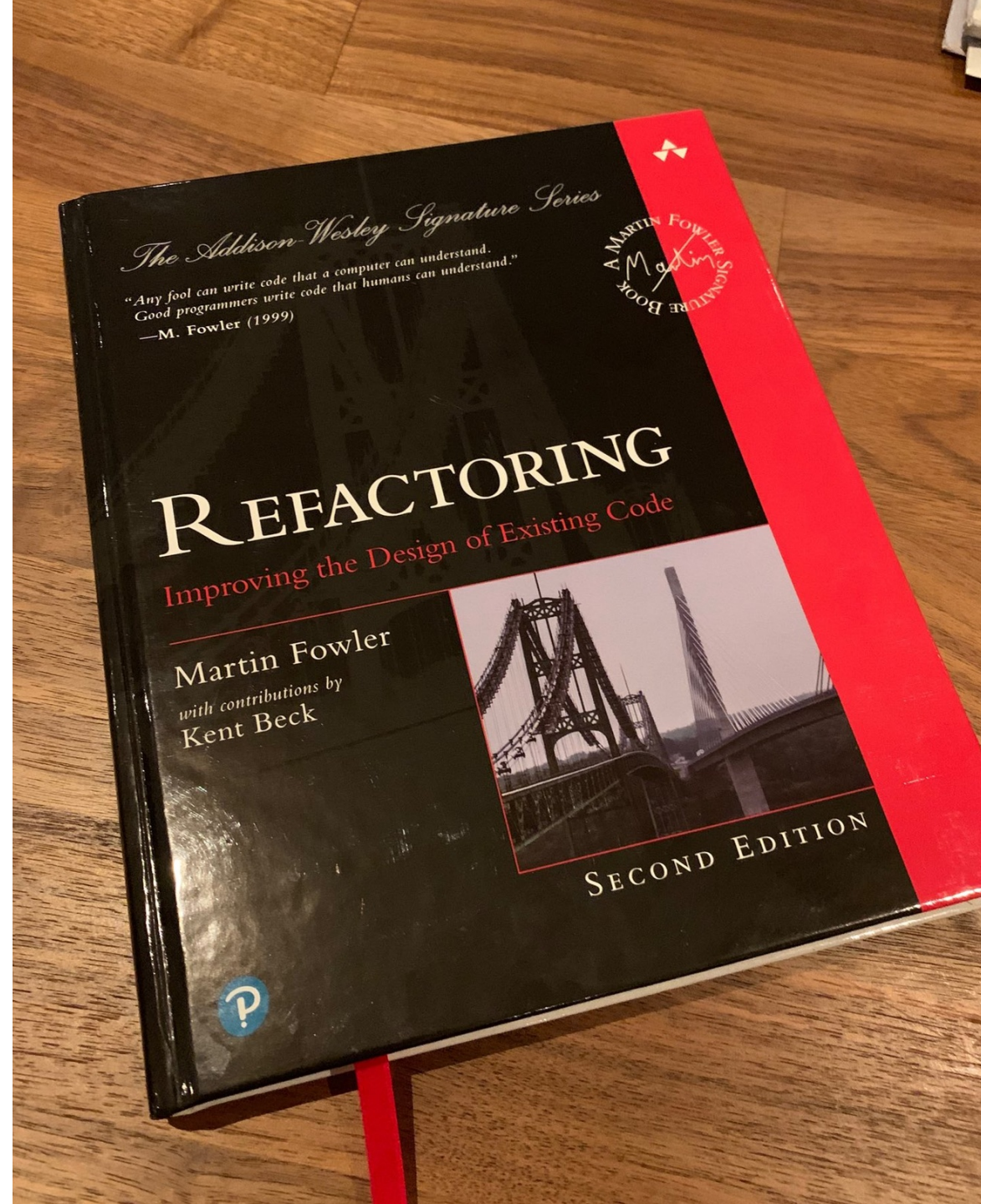
✓ Prevents change

✓ Has well named taxonomy

✓ Has a recipe to resolve

What is refactoring?

- Changing the code without changing its behaviour
- **Martin Fowler: Refactoring**
Improving the Design of Existing Code



Clean Architecture & Technical Debt at PyData London 2022

Tech Debt vs Tech Mess

- ✗ Not lack of specification, documentation, infrastructure
 - ✗ Not missing features
 - ✗ Not broken models
E.g. bias, drift, etc.
 - ✗ Not bad practice
That's tech mess
- ✓ Tech debt is an attempt to gain knowledge with a plan to correct it later.



- Motivation

—— We are here ——

- Readability

- Code Smells

- Establishing a culture

- Takeaways

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     # will store the total
3     temp = 0
4     k = 0
5     if threshold is not None and values is not None:
6         for value in values:
7             if value > threshold:
8                 temp += value
9                 k += 1
10        if k == 0:
11            return None
12        else:
13            average = temp / k
14            return average
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21    return None
```



Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     # will store the total
3     temp = 0
4     k = 0
5     if threshold is not None and values is not None:
6         for value in values:
7             if value > threshold:
8                 temp += value
9                 k += 1
10        if k == 0:
11            return None
12        else:
13            average = temp / k
14            return average
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21    return None
```

- Dead and unreachable code

•

•

•

•

•

•

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     # will store the total
3     temp = 0
4     k = 0
5     if threshold is not None and values is not None:
6         for value in values:
7             if value > threshold:
8                 temp += value
9                 k += 1
10        if k == 0:
11            return None
12        else:
13            average = temp / k
14            return average
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21    return None
```

- Dead and unreachable code

•

•

•


•

•

•

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     # will store the total
3     temp = 0
4     k = 0
5     if threshold is not None and values is not None:
6         for value in values:
7             if value > threshold:
8                 temp += value
9                 k += 1
10        if k == 0:
11            return None
12        else:
13            average = temp / k
14            return average
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21
```



- **Dead and unreachable code**

Delete code



Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     # will store the total
3     temp = 0
4     k = 0
5     if threshold is not None and values is not None:
6         for value in values:
7             if value > threshold:
8                 temp += value
9                 k += 1
10        if k == 0:
11            return None
12        else:
13            average = temp / k
14            return average
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

-

-

-

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     # will store the total ←
3     temp = 0
4     k = 0
5     if threshold is not None and values is not None:
6         for value in values:
7             if value > threshold:
8                 temp += value
9                 k += 1
10        if k == 0:
11            return None
12        else:
13            average = temp / k
14            return average
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

-

-

-

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     total = 0
3     k = 0
4     if threshold is not None and values is not None:
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            average = total / k
13            return average
14    elif values is None:
15        raise ValueError('List is None')
16    elif len(values) == 0:
17        raise ValueError('Empty list')
18    else:
19        raise ValueError('Threshold is None')
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

-

-

-

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     total = 0
3     k = 0
4     if threshold is not None and values is not None:
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            average = total / k
13            return average
14    elif values is None:
15        raise ValueError('List is None')
16    elif len(values) == 0:
17        raise ValueError('Empty list')
18    else:
19        raise ValueError('Threshold is None')
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

-


-

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     total = 0
3     k = 0
4     if threshold is not None and values is not None:
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            average = total / k
13            return average
14    elif values is None:
15        raise ValueError('List is None')
16    elif len(values) == 0:
17        raise ValueError('Empty list')
18    else:
19        raise ValueError('Threshold is None')
20
21
```



- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**



Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     total = 0
3     k = 0
4     if threshold is not None and values is not None:
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            return total / k ←
13    elif values is None:
14        raise ValueError('List is None')
15    elif len(values) == 0:
16        raise ValueError('Empty list')
17    else:
18        raise ValueError('Threshold is None')
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions



Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     total = 0
3     k = 0
4     if threshold is not None and values is not None:
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            return total / k
13    elif values is None:
14        raise ValueError('List is None')
15    elif len(values) == 0:
16        raise ValueError('Empty list')
17    else:
18        raise ValueError('Threshold is None')
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions

- **Improper variable scoping**

-

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     total = 0
3     k = 0
4     if threshold is not None and values is not None:
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            return total / k
13    elif values is None:
14        raise ValueError('List is None')
15    elif len(values) == 0:
16        raise ValueError('Empty list')
17    else:
18        raise ValueError('Threshold is None')
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions



- **Improper variable scoping**

-

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     
3
4     if threshold is not None and values is not None:
5         total = 0
6         k = 0 
7         for value in values:
8             if value > threshold:
9                 total += value
10                k += 1
11        if k == 0:
12            return None
13        else:
14            return total / k
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions

- **Improper variable scoping**

Move lines with the same variables together



Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is not None and values is not None:
3         total = 0
4         k = 0
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            return total / k
13    elif values is None:
14        raise ValueError('List is None')
15    elif len(values) == 0:
16        raise ValueError('Empty list')
17    else:
18        raise ValueError('Threshold is None')
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions

- **Improper variable scoping**

Move lines with the same variables together

- **Too many levels: if statements**

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is not None and values is not None:
3         total = 0
4         k = 0
5         for value in values:
6             if value > threshold:
7                 total += value
8                 k += 1
9         if k == 0:
10            return None
11        else:
12            return total / k
13    elif values is None:
14        raise ValueError('List is None')
15    elif len(values) == 0:
16        raise ValueError('Empty list')
17    else:
18        raise ValueError('Threshold is None')
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions

- **Improper variable scoping**

Move lines with the same variables together

- **Too many levels: if statements**

-

-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     total = 0
9     k = 0
10    for value in values:
11        if value > threshold:
12            total += value
13            k += 1
14    if k == 0:
15        return None
16    else:
17        return total / k
18
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions

- **Improper variable scoping**

Move lines with the same variables together

- **Too many levels: if statements**

Extract guard clauses



Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     total = 0
9     k = 0
10    for value in values:
11        if value > threshold:
12            total += value
13            k += 1
14    if k == 0:
15        return None
16    else:
17        return total / k
18
19
20
21
```

- **Dead and unreachable code**
Delete code
- **Comments explaining code**
Delete comments, rename variables, extract helpers
- **Excess variables**
Inline, comprehensions
- **Improper variable scoping**
Move lines with the same variables together
- **Too many levels: if statements**
Extract guard clauses
- **Too many levels: for loops**
-

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     total = 0
9     k = 0
10    for value in values:
11        if value > threshold:
12            total += value
13            k += 1
14    if k == 0:
15        return None
16    else:
17        return total / k
18
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions

- **Improper variable scoping**

Move lines with the same variables together

- **Too many levels: if statements**


Extract guard clauses

- **Too many levels: for loops**



Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     relevant_values = [v for v in values if v > threshold]
9     if len(relevant_values) == 0:
10        return None
11    else:
12        return sum(relevant_values) / len(relevant_values)
13
14
15
16
17
18
19
20
21
```



- **Dead and unreachable code**
Delete code
- **Comments explaining code**
Delete comments, rename variables, extract helpers
- **Excess variables**
Inline, comprehensions
- **Improper variable scoping**
Move lines with the same variables together
- **Too many levels: if statements**
Extract guard clauses
- **Too many levels: for loops**
Use comprehensions
-


Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     relevant_values = [v for v in values if v > threshold]
9     if len(relevant_values) == 0:
10        return None
11    else:
12        return sum(relevant_values) / len(relevant_values)
13
14
15
16
17
18
19
20
21
```

- **Dead and unreachable code**
Delete code
- **Comments explaining code**
Delete comments, rename variables, extract helpers
- **Excess variables**
Inline, comprehensions
- **Improper variable scoping**
Move lines with the same variables together
- **Too many levels: if statements**
Extract guard clauses
- **Too many levels: for loops**
Use comprehensions
- **Multiple returns**

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     relevant_values = [v for v in values if v > threshold]
9     if len(relevant_values) == 0:
10        return None
11    else:
12        return sum(relevant_values) / len(relevant_values)
13
14
15
16
17
18
19
20
21
```



- **Dead and unreachable code**
Delete code
- **Comments explaining code**
Delete comments, rename variables, extract helpers
- **Excess variables**
Inline, comprehensions
- **Improper variable scoping**
Move lines with the same variables together
- **Too many levels: if statements**
Extract guard clauses
- **Too many levels: for loops**
Use comprehensions
- **Multiple returns**

Readability

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     relevant_values = [v for v in values if v > threshold]
9     total = sum(relevant_values)
10    length = len(relevant_values)
11    return total / length if length > 0 else None
12
13
14
15
16
17
18
19
20
21
```

- **Dead and unreachable code**

Delete code

- **Comments explaining code**

Delete comments, rename variables, extract helpers

- **Excess variables**

Inline, comprehensions

- **Improper variable scoping**

Move lines with the same variables together

- **Too many levels: if statements**

Extract guard clauses

- **Too many levels: for loops**

Use comprehensions

- **Multiple returns**

Extract variables and return expression

Readability: Before - After

```
1 def calculate_average_above_threshold(values, threshold):
2     # will store the total
3     temp = 0
4     k = 0
5     if threshold is not None and values is not None:
6         for value in values:
7             if value > threshold:
8                 temp += value
9                 k += 1
10    if k == 0:
11        return None
12    else:
13        average = temp / k
14        return average
15    elif values is None:
16        raise ValueError('List is None')
17    elif len(values) == 0:
18        raise ValueError('Empty list')
19    else:
20        raise ValueError('Threshold is None')
21    return None
```

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     relevant_values = [v for v in values if v > threshold]
9     total = sum(relevant_values)
10    length = len(relevant_values)
11    return total / length if length > 0 else None
12
13
14
15
16
17
18
19
20
21
```

Readability: Outcomes

```
1 def calculate_average_above_threshold(values, threshold):
2     if threshold is None:
3         raise ValueError('Threshold is None')
4     if values is None:
5         raise ValueError('List is None')
6     if len(values) == 0:
7         raise ValueError('Empty list')
8     relevant_values = [v for v in values if v > threshold]
9     total = sum(relevant_values)
10    length = len(relevant_values)
11    return total / length if length > 0 else None
12
13
14
15
16
17
18
19
20
21
```

- The code is in one continuous logical flow
- Guard clauses on the top
- "Happy path" on the left
- Variable lifecycle is contained
- Return in the last line

- Motivation
- Readability
 - We are here —
- Code Smells
- Establishing a culture
- Takeaways

Code Smells

```
1
2
3
4 def get_active_users(
5     user_ids, user_names, times, item_ids, amounts, values,
6     cutoff_time, threshold, save_to_file, filename):
7     totals = {}
8     for user_id, time, value in zip(user_ids, times, values):
9         if user_id not in totals:
10             totals[user_id] = 0
11             if time > cutoff_time:
12                 totals[user_id] += value
13     result = {}
14     for user_id, total in totals.items():
15         if total > threshold:
16             result[user_names[user_id]] = total
17             if save_to_file:
18                 with open(filename, 'at') as outfile:
19                     outfile.write(f'{user_id}, {user_names[user_id]}\n')
20         else:
21             print(f'{user_id}, {user_names[user_id]}\n')
22     return result
23
24
25
26
```


Code Smells: Bloaters

```
1
2
3
4 def get_active_users(
5     user_ids, user_names, times, item_ids, amounts, values,
6     cutoff_time, threshold, save_to_file, filename):
7     totals = {}
8     for user_id, time, value in zip(user_ids, times, values):
9         if user_id not in totals:
10             totals[user_id] = 0
11             if time > cutoff_time:
12                 totals[user_id] += value
13     result = {}
14     for user_id, total in totals.items():
15         if total > threshold:
16             result[user_names[user_id]] = total
17             if save_to_file:
18                 with open(filename, 'at') as outfile:
19                     outfile.write(f'{user_id}, {user_names[user_id]}\n')
20         else:
21             print(f'{user_id}, {user_names[user_id]}\n')
22     return result
23
24
25
26
```

● Bloaters

Long parameter list

Data clumps

Primitive obsession



Code Smells: Bloaters

```
1
2
3
4 def get_active_users(
5     user_ids, user_names, times, item_ids, amounts, values,
6     cutoff_time, threshold, save_to_file, filename):
7     totals = {}
8     for user_id, time, value in zip(user_ids, times, values):
9         if user_id not in totals:
10             totals[user_id] = 0
11             if time > cutoff_time:
12                 totals[user_id] += value
13     result = {}
14     for user_id, total in totals.items():
15         if total > threshold:
16             result[user_names[user_id]] = total
17             if save_to_file:
18                 with open(filename, 'at') as outfile:
19                     outfile.write(f'{user_id}, {user_names[user_id]}\n')
20             else:
21                 print(f'{user_id}, {user_names[user_id]}\n')
22     return result
23
24
25
26
```

• Bloaters

Long parameter list

Data clumps

Primitive obsession

•

•

•

Code Smells: Bloaters → Extract class

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
13     active_users = []
14     for user in users:
15         total = 0
16         for transaction in user.transactions:
17             if transaction.time > cutoff_time:
18                 total += transaction.value
19         if total > threshold:
20             active_users.append(user)
21             if save_to_file:
22                 with open(filename, 'at') as outfile:
23                     outfile.write(f'{user.id}, {user.name}\n')
24             else:
25                 print(f'{user.id}, {user.name}\n')
26     return active_users
```

- **Bloaters → Extract class**

- Long parameter list

- Data clumps

- Primitive obsession

-

-

-

Code Smells: Couplers

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
13     active_users = []
14     for user in users:
15         total = 0
16         for transaction in user.transactions:
17             if transaction.time > cutoff_time:
18                 total += transaction.value
19         if total > threshold:
20             active_users.append(user)
21             if save_to_file:
22                 with open(filename, 'at') as outfile:
23                     outfile.write(f'{user.id}, {user.name}\n')
24         else:
25             print(f'{user.id}, {user.name}\n')
26     return active_users
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers**

Feature envy



Code Smells: Couplers

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
13     active_users = []
14     for user in users:
15         total = 0
16         for transaction in user.transactions:
17             if transaction.time > cutoff_time:
18                 total += transaction.value
19         if total > threshold:
20             active_users.append(user)
21             if save_to_file:
22                 with open(filename, 'at') as outfile:
23                     outfile.write(f'{user.id}, {user.name}\n')
24             else:
25                 print(f'{user.id}, {user.name}\n')
26     return active_users
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers**

Feature envy



Code Smells: Couplers → Extract method

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time): ←
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
16     active_users = []
17     for user in users:
18         if user.get_total_since(cutoff_time) > threshold: ←
19             active_users.append(user)
20             if save_to_file:
21                 with open(filename, 'at') as outfile:
22                     outfile.write(f'{user.id}, {user.name}\n')
23             else:
24                 print(f'{user.id}, {user.name}\n')
25     return active_users
26
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy



Code Smells: Couplers → Extract method 2

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time):
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
16     active_users = []
17     for user in users:
18         if user.get_total_since(cutoff_time) > threshold:
19             active_users.append(user)
20             if save_to_file:
21                 with open(filename, 'at') as outfile:
22                     outfile.write(f'{user.id}, {user.name}\n')
23             else:
24                 print(f'{user.id}, {user.name}\n')
25     return active_users
26
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy



Code Smells: Couplers → Extract method 2

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time):
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15    def to_line(self):
16        return f'{self.id}, {self.name}\n'
17
18 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
19     active_users = []
20     for user in users:
21         if user.get_total_since(cutoff_time) > threshold:
22             active_users.append(user)
23             if save_to_file:
24                 with open(filename, 'at') as outfile:
25                     outfile.write(user.to_line())
26             else:
27                 print(user.to_line())
28     return active_users
```

- **Bloaters → Extract class**

- Long parameter list
- Data clumps
- Primitive obsession

- **Couplers → Extract method**

- Feature envy



Refactor

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time):
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15    def to_line(self):
16        return f'{self.id}, {self.name}\n'
17
18 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
19     active_users = []
20     for user in users:
21         if user.get_total_since(cutoff_time) > threshold:
22             active_users.append(user)
23             if save_to_file:
24                 with open(filename, 'at') as outfile:
25                     outfile.write(user.to_line())
26             else:
27                 print(user.to_line())
28     return active_users
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy



Refactor

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time):
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15    def to_line(self):
16        return f'{self.id}, {self.name}\n'
17
18 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
19     active_users = []
20     for user in users:
21         if user.get_total_since(cutoff_time) > threshold:
22             active_users.append(user)
23             if save_to_file:
24                 with open(filename, 'at') as outfile:
25                     outfile.write(user.to_line())
26             else:
27                 print(user.to_line())
28     return active_users
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy



Refactor

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time):
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15    def to_line(self):
16        return f'{self.id}, {self.name}\n'
17
18 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
19     active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
20     if save_to_file:
21         with open(filename, 'wt') as outfile:
22             for user in active_users:
23                 outfile.write(user.to_line())
24     else:
25         for user in active_users:
26             print(user.to_line())
27     return active_user
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy

-

-

Boolean parameters

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time):
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15    def to_line(self):
16        return f'{self.id}, {self.name}\n'
17
18 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
19     active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
20     if save_to_file:
21         with open(filename, 'wt') as outfile:
22             for user in active_users:
23                 outfile.write(user.to_line())
24     else:
25         for user in active_users:
26             print(user.to_line())
27     return active_user
```

- **Bloaters → Extract class**
 - Long parameter list
 - Data clumps
 - Primitive obsession
- **Couplers → Extract method**
 - Feature envy
- **Boolean parameters**



Boolean parameters

```
1 @dataclass
2 class Transaction:
3     time: datetime
4     value: int
5
6 @dataclass
7 class User:
8     id: int
9     name: str
10    transactions: List[Transaction]
11
12    def get_total_since(self, cutoff_time):
13        return sum([t.value for t in self.transactions if t.time > cutoff_time])
14
15    def to_line(self):
16        return f'{self.id}, {self.name}\n'
17
18 def get_active_users(users, cutoff_time, threshold, save_to_file, filename):
19     active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
20     if save_to_file:
21         with open(filename, 'wt') as outfile:
22             for user in active_users:
23                 outfile.write(user.to_line())
24     else:
25         for user in active_users:
26             print(user.to_line())
27     return active_user
```

- **Bloaters → Extract class**
 - Long parameter list
 - Data clumps
 - Primitive obsession
- **Couplers → Extract method**
 - Feature envy
- **Boolean parameters**

Boolean parameters → Dependency Injection

```
1 class ActiveUserService:
2
3     def __init__(self):
4         pass
5
6     def get(self, users, cutoff_time, threshold, save_to_file, filename):
7         active_users = [u for u in users if u.get_total_since(cutoff_time) > t
8         if save_to_file:
9             with open(filename, 'wt') as outfile:
10                 for user in active_users:
11                     outfile.write(user.to_line())
12         else:
13             for user in active_users:
14                 print(user.to_line())
15         return active_users
16
17 def main(cutoff_time, threshold, save_to_file, filename):
18     users = ... # loads users
19     active_user_service = ActiveUserService()
20     active_users = active_user_service.get(
21         users=users,
22         cutoff_time=cutoff_time,
23         threshold=threshold,
24         save_to_file=save_to_file,
25         filename=filename
26     )
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy

- **Boolean parameters → Dependency Injection**

Move code to class



Boolean parameters → Dependency Injection

```
1 class ActiveUserService:
2
3     def __init__(self):
4         pass
5
6     def get(self, users, cutoff_time, threshold, save_to_file, filename):
7         active_users = [u for u in users if u.get_total_since(cutoff_time) > t
8         if save_to_file:
9             with open(filename, 'wt') as outfile:
10                 for user in active_users:
11                     outfile.write(user.to_line())
12
13         else:
14             for user in active_users:
15                 print(user.to_line())
16         return active_users
17
18 def main(cutoff_time, threshold, save_to_file, filename):
19     users = ... # loads users
20     active_user_service = ActiveUserService()
21     active_users = active_user_service.get(
22         users=users,
23         cutoff_time=cutoff_time,
24         threshold=threshold,
25         save_to_file=save_to_file,
26         filename=filename
27     )
```

- **Bloaters → Extract class**
 - Long parameter list
 - Data clumps
 - Primitive obsession
- **Couplers → Extract method**
 - Feature envy
- **Boolean parameters → Dependency Injection**
 - Move code to class
 - Instantiate and call it in main()

Boolean parameters → Dependency Injection

```
1 class FileOutput:
2     def __init__(self, filename):
3         self.filename = filename
4
5     def output(self, users):
6         with open(self.filename, 'wt') as outfile:
7             for user in users:
8                 outfile.write(user.to_line())
9
10 class ActiveUserService:
11
12     def __init__(self, output):
13         self.output = output
14
15     def get(self, users, cutoff_time, threshold):
16         active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
17         self.output.output(active_users)
18         return active_users
19
20 def main(cutoff_time, threshold, filename):
21     users = ... # loads users
22     active_user_service = ActiveUserService(output=FileOutput(filename))
23     active_users = active_user_service.get(
24         users=users,
25         cutoff_time=cutoff_time,
26         threshold=threshold,
27     )
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy

- **Boolean parameters → Dependency Injection**

Move code to class

Instantiate and call it in main()

Extract conditional code into Strategy Pattern



Boolean parameters → Dependency Injection

```
1 class FileOutput:
2     def __init__(self, filename):
3         self.filename = filename
4
5     def output(self, users):
6         with open(self.filename, 'wt') as outfile:
7             for user in users:
8                 outfile.write(user.to_line())
9
10 class ActiveUserService:
11
12     def __init__(self, output):
13         self.output = output
14
15     def get(self, users, cutoff_time, threshold):
16         active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
17         self.output.output(active_users)
18         return active_users
19
20 def main(cutoff_time, threshold, filename):
21     users = ... # loads users
22     active_user_service = ActiveUserService(output=FileOutput(filename))
23     active_users = active_user_service.get(
24         users=users,
25         cutoff_time=cutoff_time,
26         threshold=threshold,
27     )
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy

- **Boolean parameters → Dependency Injection**

Move code to class

Instantiate and call it in main()

Extract conditional code into Strategy Pattern

Inject strategy into the class



Boolean parameters → Alternative options

```
1
2
3 class PrintOutput:
4     def output(self, users):
5         for user in users:
6             print(user.to_line())
7
8 class ActiveUserService:
9
10    def __init__(self, output):
11        self.output = output
12
13    def get(self, users, cutoff_time, threshold):
14        active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
15        self.output.output(active_users)
16        return active_users
17
18 def main(cutoff_time, threshold, filename):
19     users = ... # loads users
20     active_user_service = ActiveUserService(output=PrintOutput())
21     active_users = active_user_service.get(
22         users=users,
23         cutoff_time=cutoff_time,
24         threshold=threshold,
25     )
26
27
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy

- **Boolean parameters → Dependency Injection**

Move code to class

Instantiate and call it in main()

Extract conditional code into Strategy Pattern

Inject strategy into the class



Boolean parameters → Alternative options

```
1
2
3 class NoOutput:
4     def output(self, users):
5         pass
6
7 class ActiveUserService:
8
9     def __init__(self, output):
10         self.output = output
11
12     def get(self, users, cutoff_time, threshold):
13         active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
14         self.output.output(active_users)
15         return active_users
16
17 def main(cutoff_time, threshold, filename):
18     users = ... # loads users
19     active_user_service = ActiveUserService(output=NoOutput())
20     active_users = active_user_service.get(
21         users=users,
22         cutoff_time=cutoff_time,
23         threshold=threshold,
24     )
25
26
27
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy

- **Boolean parameters → Dependency Injection**

Move code to class

Instantiate and call it in main()

Extract conditional code into Strategy Pattern

Inject strategy into the class



Boolean parameters → Alternative options

```
1 class BothOutput:
2     def __init__(self, outputs):
3         self.outputs = outputs
4
5     def output(self, users):
6         for output in self.outputs:
7             output.output(users)
8
9 class ActiveUserService:
10
11     def __init__(self, output):
12         self.output = output
13
14     def get(self, users, cutoff_time, threshold):
15         active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
16         self.output.output(active_users)
17         return active_users
18
19 def main(cutoff_time, threshold, filename):
20     users = ... # loads users
21     active_user_service = ActiveUserService(output=BothOutput(
22         outputs=[FileOutput(filename), PrintOutput()]
23     ))
24     active_users = active_user_service.get(
25         users=users,
26         cutoff_time=cutoff_time,
27         threshold=threshold,
28     )
```

- **Bloaters → Extract class**

Long parameter list

Data clumps

Primitive obsession

- **Couplers → Extract method**

Feature envy

- **Boolean parameters → Dependency Injection**

Move code to class

Instantiate and call it in main()

Extract conditional code into Strategy Pattern

Inject strategy into the class



Code Smells: Couplers

```
1 class BothOutput:
2     def __init__(self, outputs):
3         self.outputs = outputs
4
5     def output(self, users):
6         for output in self.outputs:
7             output.output(users)
8
9 class ActiveUserService:
10
11     def __init__(self, output):
12         self.output = output
13
14     def get(self, users, cutoff_time, threshold):
15         active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
16         self.output.output(active_users)
17         return active_users
18
19 def main(cutoff_time, threshold, filename):
20     users = ... # loads users
21     active_user_service = ActiveUserService(output=BothOutput(
22         outputs=[FileOutput(filename), PrintOutput()]
23     ))
24     active_users = active_user_service.get(
25         users=users,
26         cutoff_time=cutoff_time,
27         threshold=threshold,
28     )
```

- **Bloaters → Extract class**
 - Long parameter list
 - Data clumps
 - Primitive obsession
- **Couplers → Extract method**
 - Feature envy
- **Boolean parameters → Dependency Injection**
- **Couplers**
 - Empty class
 - Middle man
 - Message chain
 - Speculative Generality

Code Smells: Couplers → Delete class

```
1
2
3
4
5
6 class BothOutput:
7     def __init__(self, outputs):
8         self.outputs = outputs
9
10    def output(self, users):
11        for output in self.outputs:
12            output.output(users)
13
14 def main(cutoff_time, threshold, filename):
15     output = BothOutput(outputs=[
16         FileOutput(filename),
17         PrintOutput(),
18     ])
19     users = ... # loads users
20     active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
21     output.output(active_users)
22     ...
23
24
25
26
27
```

- **Bloaters → Extract class**
 - Long parameter list
 - Data clumps
 - Primitive obsession
- **Couplers → Extract method**
 - Feature envy
- **Boolean parameter → Dependency Injection**
- **Couplers → Delete class**
 - Empty class
 - Middle man
 - Message chain
 - Speculative Generality

Improved Refactoring: Before - After

```
1
2
3
4
5 def get_active_users(
6     user_ids, user_names, times, item_ids, amounts, values,
7     cutoff_time, threshold, save_to_file, filename):
8     totals = {}
9     for user_id, time, value in zip(user_ids, times, values):
10         if user_id not in totals:
11             totals[user_id] = 0
12         if time > cutoff_time:
13             totals[user_id] += value
14     result = {}
15     for user_id, total in totals.items():
16         if total > threshold:
17             result[user_names[user_id]] = total
18             if save_to_file:
19                 with open(filename, 'wt') as outfile:
20                     outfile.write(f'{user_id}, {user_names[user_id]}\n')
21             else:
22                 print(f'{user_id}, {user_names[user_id]}\n')
23     return result
24
25
26
27
```

```
1 @dataclass
2 class User:
3     id: int
4     name: str
5     transactions: List[Transaction]
6
7     def get_total_since(self, cutoff_time):
8         return sum([t.value for t in self.transactions if t.time > cutoff_time])
9
10    def to_line(self):
11        return f'{self.id}, {self.name}\n'
12
13 class FileOutput:
14     def __init__(self, filename):
15         self.filename = filename
16
17     def output(self, users):
18         with open(self.filename, 'wt') as outfile:
19             for user in users:
20                 outfile.write(user.to_line())
21
22 def main(cutoff_time, threshold, filename):
23     output = FileOutput(filename)
24     users = ... # loads users
25     active_users = [u for u in users if u.get_total_since(cutoff_time) > threshold]
26     output.output(active_users)
27     ...
```

- Motivation

- Readability

- Code Smells

— We are here —

- Establishing a culture

- Takeaways

Establishing a culture

- **Code review**
Programming is communication
- **Total cost of ownership**
Manage long and short term goals
- **Developer happiness**
Autonomy - Mastery - Relatedness
- **Drive cultural change**
Increased velocity

Takeaways

- Programming is communication
- Concentrate on the dataflow
- Identify problem areas
- Prepare the code
- Identify code smells
- Use refactoring recipes
- Evaluate the TCO of code
- Establish a culture
- Blog: laszlo.substack.com
- Community: cq4ds.com