Program Design

Invasion Percolation: Testing

Program Design

Invasion Percolation: ~~Testing~~

We found one bug

We found one bug

How many others *haven't* we found?

We found one bug

How many others *haven't* we found?

How do we *validate* and *verify* this program?

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

- Validation: are we using a good model?

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

- Validation: are we using a good model?

The second question is a question for scientists...

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

- Validation: are we using a good model?

The second question is a question for scientists...

...so we'll concentrate on testing our program

We found one bug

How many others *haven't* we found?

How do we *verify* and *validate* this program?

- Verification: is our program free of bugs?

- Validation: are we using a good model?

The second question is a question for scientists...

...so we'll concentrate on ~~testing our program~~

making our program testable

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid…

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
−1	−1	−1	2	2
2	2	2	2	2
2	2	2	2	2

...should fill in like this

2	2	2	2	2
2	2	2	2	2
1	1	1	2	2
2	2	2	2	2
2	2	2	2	2

This grid...

2	2	2	2	2
2	2	2	2	2
−1	−1	−1	2	2
2	2	2	2	2
2	2	2	2	2

...should fill in like this

If it doesn't, it should be easy to figure out why not

Overall program structure

```
'''doc string'''
def fail(...): ...
def create_random_grid(N, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
if __name__ == '__main__':
  ...
```

Overall program structure

```
'''doc string'''
def fail(...): ...
def create_random_grid(N, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
if __name__ == '__main__':
  ...
```

This is what we want to test

Overall program structure

```
'''doc string'''
def fail(...): ...
def create_random_grid(N, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
if __name__ == '__main__':
  ...
```

This is what we want to test

Let's reorganize the code so that it's easy to create specific grids

Old structure

```
...
def create_random_grid(N, Z): ...
...
if __name__ == '__main__':
  ...
  grid = create_random_grid(grid_size, value_range)
  ...
```

New structure

```
...
def create_grid(N): ...
def fill_grid_random(grid, Z): ...
...
if __name__ == '__main__':
  ...
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  ...
```

New structure

```
...
def create_grid(N): ...
def fill_grid_random(grid, Z): ...
...
if __name__ == '__main__':
  ...
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  ...
```

Create an N×N grid of zeroes

New structure

```
...
def create_grid(N): ...
def fill_grid_random(grid, Z): ...
...
if __name__ == '__main__':
  ...
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  ...
```

Create an N×N grid of zeroes

Overwrite cells with random values in 1..Z

New structure

```
...
def create_grid(N): ...
def fill_grid_random(grid, Z): ...
...
if __name__ == '__main__':
  ...
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  ...
```

Create an N×N grid of zeroes

Overwrite cells with random values in 1..Z

We'll call something else when testing

Old structure

```
...
if __name__ == '__main__':
  # Get parameters from command line.
  arguments = sys.argv[1:]
  try:
    grid_size = int(arguments[0])
    value_range = int(arguments[1])
    rand_seed = int(arguments[2])
  except ...:
    ...
```

New structure

```
...
if __name__ == '__main__':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[1:])
    ...
```

Newer structure

```
...
if __name__ == '__main__':
  scenario = sys.argv[1]
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  if scenario == 'random':
    ...
```

Newer structure

```
...
if __name__ == '__main__':
  scenario = sys.argv[1]
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  if scenario == 'random':
    ...
  else:
    fail('Unknown scenario %s' % scenario)
```

We aren't going to need random numbers when we fill the grid manually for testing

We aren't going to need random numbers when we fill the grid manually for testing

We're also not going to need the value range

We aren't going to need random numbers

in cases where we fill the grid manually for testing

We're also not going to need the value range

Or the grid size

We aren't going to need random numbers

in cases where we fill the grid manually for testing

We're also not going to need the value range

Or the grid size

So move argument handling and RNG seeding

into the random scenario

The revised structure

```
...
if __name__ == '__main__':
  scenario = sys.argv[1]
  if scenario == 'random':
    grid_size, value_range, rand_seed = \
      parse_arguments(sys.argv[2:])
    random.seed(rand_seed)
    ...
  else:
    fail('Unknown scenario %s' % scenario)
```

The revised structure

```
...
if __name__ == '__main__':
  scenario = sys.argv[1]
  if scenario == 'random':
    grid_size, value_range, rand_seed = \
      parse_arguments(sys.argv[2:])
    random.seed(rand_seed)
    ...
  else:
    fail('Unknown scenario %s' % scenario)
```

The revised structure

```
...
if __name__ == '__main__':
  scenario = sys.argv[1]
  if scenario == 'random':
    grid_size, value_range, rand_seed = \
      parse_arguments(sys.argv[2:])
    random.seed(rand_seed)
    ...
  else:
    fail('Unknown scenario %s' % scenario)
```

The revised structure

```
...
if __name__ == '__main__':
  scenario = sys.argv[1]
  if scenario == 'random':
    grid_size, value_range, rand_seed = \
      parse_arguments(sys.argv[2:])
    random.seed(rand_seed)
    ...
  else:
    fail('Unknown scenario %s' % scenario)
```

The revised structure

```
...
if __name__ == '__main__':
  scenario = sys.argv[1]
  if scenario == 'random':
    grid_size, value_range, rand_seed = \
      parse_arguments(sys.argv[2:])
    random.seed(rand_seed)
    ...
  else:
    fail('Unknown scenario %s' % scenario)
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

A closer look

```
if scenario == 'random':
  grid_size, value_range, rand_seed = \
    parse_arguments(sys.argv[2:])
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

```
if scenario == 'random':
  ...get arguments...
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  fill_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

These function names are too similar

So rename

```
if scenario == 'random':
  ...get arguments...
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  init_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

```
if scenario == 'random':
  ...get arguments...
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  init_grid_random(grid, value_range)
  mark_filled(grid, grid_size/2, grid_size/2)
  num_filled_cells = fill_grid(grid) + 1
  print '%d cells filled' % num_filled_cells
```

And since we do this

in all scenarios...

```
if scenario == 'random':
  ...get arguments...
  random.seed(rand_seed)
  grid = create_grid(grid_size)
  init_grid_random(grid, value_range)
  num_filled_cells = fill_grid(grid)
  print '%d cells filled' % num_filled_cells
```

...move it into

fill_grid

Reminder of revised program structure

```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments(arguments): ...
if __name__ == '__main__':
  ...
```

```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments(arguments): ...
if __name__ == '__main__':
  ...
```

Created by splitting a function that was doing two things

```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments(arguments): ...
if __name__ == '__main__':
  ...
```

Fills middle cell at the start, and returns count of *all* filled cells

```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments(arguments): ...
if __name__ == '__main__':
  ...
```

Handle arguments

```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments(arguments): ...
if __name__ == '__main__':
  ...
```

Handle arguments

for random case

```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments(arguments): ...
if __name__ == '__main__':
  ...
```

Handle arguments for random case

We should rename it to make that clear...

```
'''doc string'''
def fail(...): ...
def create_grid(N): ...
def init_grid_random(grid, Z): ...
def mark_filled(grid, x, y): ...
def is_candidate(grid, x, y): ...
def find_candidates(grid): ...
def fill_grid(grid): ...
def parse_arguments_random(arguments): ...
if __name__ == '__main__':
  ...
```

We set out to test our program...

We set out to test our program...

...but found we had to reorganize it first

We set out to test our program...

...but found we had to ~~reorganize~~ it first

refactor

We set out to test our program...

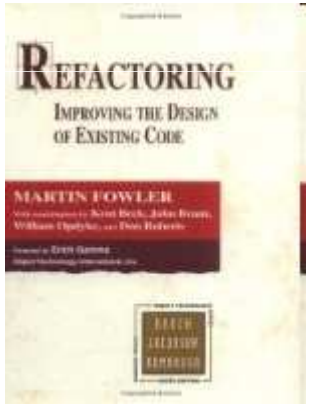
...but found we had to ~~reorganize~~ it first

refactor

Refactoring: changing a program's structure without modifying its behavior or functionality in order to improve its quality

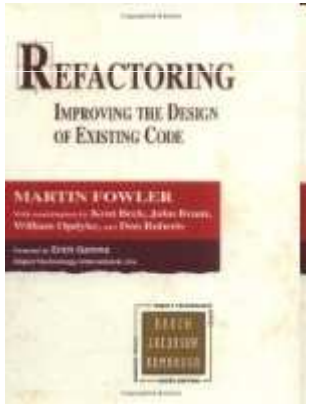
There are whole books about how to do this systematically

There are whole books about how to do this systematically



The original

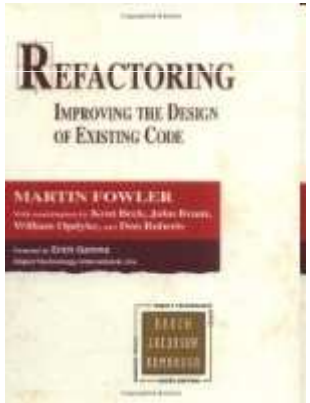
There are whole books about how to do this systematically



The original

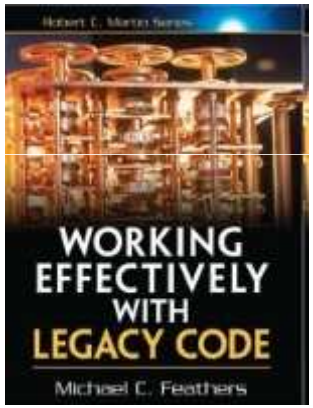
Mostly a catalog of refactorings for object-oriented programs

There are whole books about how to do this systematically



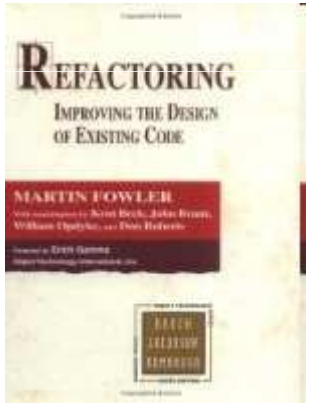
The original

Mostly a catalog of refactorings for object-oriented programs



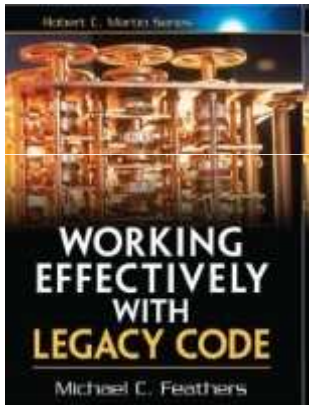
How to refactor legacy programs to make them more testable

There are whole books about how to do this systematically



The original

Mostly a catalog of refactorings for object-oriented programs



How to refactor legacy programs to make them more testable

Examples drawn from many languages

And *now* we can start testing our program

software carpentry

created by

Greg Wilson

May 2010