

How AI Coding Assistants Improve Code Habitability

Paul Sobocinski | spikes.sobes.co

 thoughtworks

How do we **feel** about
the code we ship today?



How do we feel about
the code we ship today?

Can we set a **higher bar**?



Today we explore...

1. Why setting a **higher bar** matters
2. How **code habitability** helps set that bar
3. How **AI coding assistants** get us there



Paul Sobocinski



20 yrs full-stack
Software Engineer



Paul Sobocinski



20 yrs full-stack
Software Engineer



Coaching Software
Engineers & Tech Leads



1.5 yrs exploring AI
coding assistants



Why setting a **higher bar** matters

 **Software**

**Developers are
starting to love
GenAI tools**

74%


can focus on
more satisfying
work

72%

Are
(very) favourable
of AI tools

32

Our internal
NPS score
for GitHub Copilot

 **With GenAI,**
it's never been
easier to
write code

88%

of devs say
they're more
productive

55%

faster task
completion with
AI Coding Assistants

81%

agree **increased**
productivity is
biggest benefit



Unprecedented
volume of code
being shipped

4.5B Total contributions
on GitHub in 2023

98M new projects
started in 2023

81% private repos **activity**
and **growth rate**
40%

80% of developers are **not happy** at their job.

Tech debt

62.4%

Build stack complexity

32.9%

Deployment stack complexity

32.3%

Tools / systems reliability

31.5%

*"Select
all that
apply"*

80% of developers are **not happy** at their job.

Tech debt 62.4%

Build stack complexity 32.9%

Deployment stack complexity 32.3%

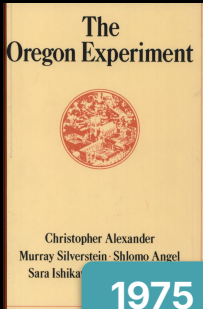
Tools / systems reliability 31.5%

*"Select
all that
apply"*

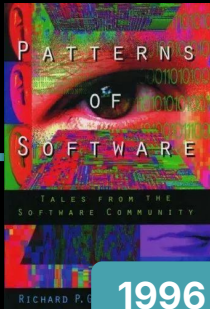


How **code habitability** helps set the bar

Habitability : Origins



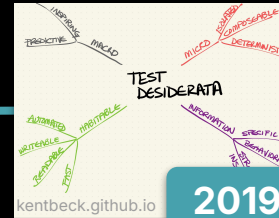
1975



1996



2017

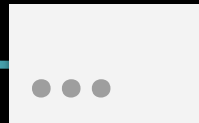


2019

YOW!

CUPID
For Joyful Coding
DANIEL TERHORST-NORTH

2022



Habitability : Definition

What it is:

The characteristic of source code that enables [people] **coming to the code later in its life** to understand its construction and intentions and to **change it comfortably and confidently**.

What it does:

Habitability **makes a [codebase] livable**, like home. And this is what we want in software — that developers feel at home, can **place their hands on any item without having to think deeply about where it is**.

Effects of Low Habitability



Breaking changes are likely



Breaking changes take longer to fix



Features take longer to add



Changes take much longer than predicted

Low Habitability

Tests are:

- Nonexistent
- Confusing
- Flakey
- Slow

Low Habitability

Tests are:

- Nonexistent
- Confusing
- Flakey
- Slow

** bi-directional, circular, unnecessary, etc.*

Implementation code has:

- Ambiguous or misleading naming
- Confusing structure or organization
- Needless complexity
- Entangled* dependencies

High Habitability

Tests:

- + Pass or fail when they are meant to
- + Express what is being tested & why
- + Communicate the code's design
- + Exhibit established design principles*

High Habitability

Tests:

- + Pass or fail when they are meant to
- + Express what is being tested & why
- + Communicate the code's design
- + Exhibit established design principles*

* *Test Desiderata, Practical Test Pyramid, Use of TDD / BDD, etc.*

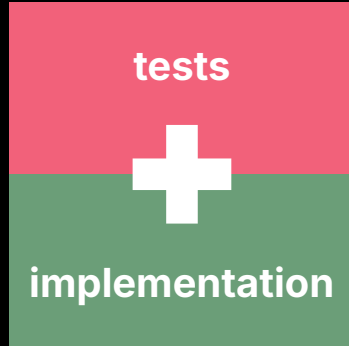
** *loose coupling, composable, dependency inversion, etc.*

*** *Four Rules of Simple Design, SOLID, CUPID, etc.*

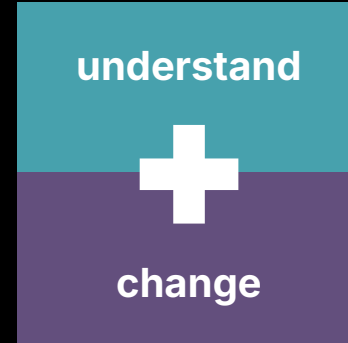
Implementation code:

- + Has intention-revealing naming and structure
- + Is as simple as possible
- + Has manageable dependencies**
- + Exhibits established*** design principles

Habitability takeaways



*Habitability applies to
both codebases*



*We can apply **both**
behaviours to the code*

How **AI coding assistants** get us there...

DEMO

README [1 of 2]

We have inherited a `URLShortener`. Two public methods exist on the class:

README [1 of 2]

We have inherited a `URLShortener`. Two public methods exist on the class:

1. `URLShortener.shorten(long_url)`
 - a. Given a full-length URL,
generate a shortened version of the URL, and
store the original URL in the DB

README [1 of 2]

We have inherited a `URLShortener`. Two public methods exist on the class:

1. `URLShortener.shorten(long_url)`
 - a. Given a full-length URL,
generate a shortened version of the URL, and
store the original URL in the DB
2. `URLShortener.retrieve(short_url)`
 - a. Given an existing shortened URL,
retrieve the full-length URL from the DB
 - b. Given a non-existent shortened URL,
raise a "not found" error

README [2 of 2]

`URLShortener` will be extended in the following way:

1. Use a better URL shortening algorithm
2. Use a more scalable DB technology

README [2 of 2]

`URLShortener` will be extended in the following way:

1. Use a better URL shortening algorithm
2. Use a more scalable DB technology

To make the above changes easier, we wish to:

1. Verify that `URLShortener` is safe to change
2. Redesign `URLShortener` so that it can be extended

```

1 require './app/url_shortener'
2 require './spec_helper'
3
4 RSpec.describe URLShortener do
5   before(:all) do
6     @db = Daybreak::DB.new './data/urls.db'
7   end
8
9   describe '.shorten' do
10    it 'shortens the url' do
11      short_url = URLShortener.shorten('https://www.yahoo.com')
12
13      expect(short_url.size).to be < 'https://www.yahoo.com'.size
14    end
15
16    it 'writes shortened url the db' do
17      short_url = URLShortener.shorten('https://www.google.com')
18
19      @db.load
20      retrieved_url = @db[short_url]
21      expect(retrieved_url).to eq 'https://www.google.com'
22    end
23  end
24
25  describe '.retrieve' do
26    it 'retrieves full-length url from the db' do
27      @db['https://short_url.com'] = 'https://www.full_length_url.com'
28
29      long_url = URLShortener.retrieve('https://short_url.com')
30
31      expect(long_url).to eq 'https://www.full_length_url.com'
32    end
33
34    it 'raises a "not found" error when url is not in the db' do
35      expect do
36        URLShortener.retrieve('https://unknown_url.com')
37      end.to raise_error('URL not found: https://unknown_url.com')
38    end
39  end
40
41  after(:all) do
42    @db.close
43    File.delete('./data/urls.db')
44  end
45 end

```

```

1 require 'daybreak'
2
3 class URLShortener
4   def self.shorten(full_length_url)
5     short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
6     db = Daybreak::DB.new './data/urls.db'
7     db.set!(short_url, full_length_url)
8     db.close
9
10    short_url
11  end
12
13  def self.retrieve(short_url)
14    db = Daybreak::DB.new './data/urls.db'
15
16    unless db.keys.include?(short_url)
17      db.close
18      raise "URL not found: #{short_url}"
19    end
20
21    full_length_url = db[short_url]
22    db.close
23
24    full_length_url
25  end
26 end

```



How can I improve the tests in `#selection`?

```
4 RSpec.describe URLShortener do
5   before(:all) do
6     @db = Daybreak::DB.new './data/urls.db'
7   end
8
9   describe '.shorten' do
10    it 'shortens the url' do
11      short_url = URLShortener.shorten('https://www.yahoo.com')
12
13      expect(short_url.size).to be < 'https://www.yahoo.com'.size
14    end
15
16    it 'writes shortened url the db' do
17      short_url = URLShortener.shorten('https://www.google.com')
18
19      @db.load
20      retrieved_url = @db[short_url]
21      expect(retrieved_url).to eq 'https://www.google.com'
22    end
23  end
24
25  describe '.retrieve' do
26    it 'retrieves full-length url from the db' do
27      @db['https://short_url.com'] = 'https://www.full_length_url.com'
28
29      long_url = URLShortener.retrieve('https://short_url.com')
30
31      expect(long_url).to eq 'https://www.full_length_url.com'
32    end
33
34    it 'raises a "not found" error when url is not in the db' do
35      expect do
36        URLShortener.retrieve('https://unknown_url.com')
37      end.to raise_error('URL not found: https://unknown_url.com')
38    end
39  end
40
41  after(:all) do
42    @db.close
43    File.delete('./data/urls.db')
44  end
45 end
```

GitHub Copilot



Isolation of tests



Test for edge cases



Use of `let` for setup
(memoized test helper)

- How can I improve the tests in **#selection?**
- How can I move in **small steps?**



```
7 end
8
9 describe '.shorten' do
10   it 'shortens the url' do
11     s
12   end
13
14   end
15
16   it 'writes shortened url the db' do
17     short_url = URLShortener.shorten('https://www.google.com')
18
19     @db.load
20     retrieved_url = @db[short_url]
21     expect(retrieved_url).to eq 'https://www.google.com'
22   end
23 end
24
25 describe '.retrieve' do
26   it 'retrieves full-length url from the db' do
27     @db['https://short_url.com'] = 'https://www.full_length_url.com'
28
29     long_url = URLShortener.retrieve('https://short_url.com')
30
31     expect(long_url).to eq 'https://www.full_length_url.com'
32   end
33
34   it 'raises a "not found" error when url is not found' do
35     expect do
36       URLShortener.retrieve('https://unknown_url.com')
37     end.to raise_error('URL not found')
38   end
39 end
40
41 after(:all) do
42   @db.close
43   File.delete('./data/urls.db')
44 end
45 end
```

Change **before (:all)** to **before (:each)**

Pass **db** as an argument

Replace **@db** with **let (:db)**

Add new tests for edge cases

Run tests after each (small) step

+ Isolation

+ Maintainability

+ Speed

~ (skip)

+ Confidence

GitHub Copilot

Used 1 reference >

Here's a step-by-step guide:

new database instance for each test.

retrieve calls in your tests to pass @db as an argument.

retrieve method definition

4. Step 4: Replace @db with let(:db). This is more idiomatic in RSpec and lazily evaluates

```
let(:db) { Daybreak::DB.new './data/urls.db' }
```

shortened URL or d

it 'raises an error when trying to shorten an invalid url' do

test code

edge cases.

7. Step 7: Run your tests after each step to ensure everything still works as expected.

Remember, the key to moving in small steps is to make one small change at a time and then test your code to ensure it still works as expected. This allows you to catch and fix errors as soon as they occur, making the debugging process much easier.

```

1 require './app/url_shortener'
2 require './spec_helper'
3
4 RSpec.describe URLShortener do
5   before(:each) do
6     @db = Daybreak::DB.new './data/urls.db'
7   end
8
9   describe '.shorten' do
10    it 'shortens the url' do
11      short_url = URLShortener.shorten('https://www.yahoo.com', @db)
12
13      expect(short_url.size).to be < 'https://www.yahoo.com'.size
14    end
15
16    it 'writes shortened url the db' do
17      short_url = URLShortener.shorten('https://www.google.com', @db)
18
19      @db.load
20      retrieved_url = @db[short_url]
21      expect(retrieved_url).to eq 'https://www.google.com'
22    end
23  end
24
25  describe '.retrieve' do
26    it 'retrieves full-length url from the db' do
27      @db['https://short_url.com'] = 'https://www.full_length_url.com'
28
29      long_url = URLShortener.retrieve('https://short_url.com', @db)
30
31      expect(long_url).to eq 'https://www.full_length_url.com'
32    end
33
34    it 'raises a "not found" error when url is not in the db' do
35      expect do
36        URLShortener.retrieve('https://unknown_url.com', @db)
37      end.to raise_error('URL not found: https://unknown_url.com')
38    end
39  end
40
41  after(:each) do
42    @db.close
43    File.delete('./data/urls.db')
44  end
45 end

```

```

1 require 'daybreak'
2
3 class URLShortener
4   def self.shorten(full_length_url, db)
5     short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
6     db.set!(short_url, full_length_url)
7     db.close
8
9     short_url
10  end

```

```

13 Run options: include {:focus=>true}
14 All examples were filtered out; ignoring {:focus=>true}
15 URLShortener
16   .shorten
17     shortens the url
18     writes shortened url the db (FAILED - 1)
19   .retrieve
20     retrieves full-length url from the db
21     raises a "not found" error when url is not in the db

```

Failures:

```

1) URLShortener.shorten writes shortened url the db
   Failure/Error: @db.load

```

```

ThreadError:
  stopping only thread
  note: use $sleep to stop forever
# ./spec/url_shortener_spec.rb:19:in `block (3 levels) in <top (required)>'

```

```

Finished in 0.41194 seconds (files took 0.11645 seconds to load)
4 examples, 1 failure

```

Failed examples:

```

rspec ./spec/url_shortener_spec.rb:16 # URLShortener.shorten writes shortened url the db
zsh returned exit code 1

```





@workspace /fix code in #selection based on error message in #terminalSelection

thoughtworks-psobocinski

@workspace /fix code in #selection based on error message in #terminalSelection

GitHub Copilot used @workspace /fix

Used 3 references

- url_shortener.rb:4-10 app
- url_shortener.rb:4-10 #selection
- #terminalSelection

The error message indicates that there's a problem with the `db.load` call, which is likely due to the database being closed

✓ **Correct diagnosis:**
@db.load called after **db.close**

```
def self.shorten(full_length_url, db = Daybreak::DB.new('./data/urls.db'))
  short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
  db.set!(short_url, full_length_url)
  short_url
end
```

In this version, `db.close` is removed. You should call `db.close` in the part of your code where you're done with all database operations. If you're using a framework like Rails, this would typically be in a controller action, after all model operations are done.

```
38     end
39   end
40
41   after(:each) do
42     @db.close
43     File.delete('./data/urls.db')
44   end
45 end
```

```
def self.shorten(full_length_url, db)
  short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
  db.set!(short_url, full_length_url)

  short_url
end
```

Run options: include {:focus=>true}

All examples were filtered out; ignoring {:focus=>true}

```
URLShortener
  .shorten
    shortens the url
    writes shortened url the db (FAILED - 1)
  .retrieve
    retrieves full-length url from the db
    raises a "not found" error when url is not in the db
```

Failures:

1) URLShortener.shorten writes shortened url the db
 Failure/Error: @db.load

```
ThreadError:
  stopping only thread
  note: use $sleep to stop forever
  # ./spec/url_shortener_spec.rb:19:in `block (3 levels) in <top (required)>
```

Finished in 0.41194 seconds (files took 0.11645 seconds to load)
 4 examples, 1 failure

Failed examples:

```
rspec ./spec/url_shortener_spec.rb:16 # URLShortener.shorten writes shortened url the db
zsh returned exit code 1
```





Propose a cleanup to the code in `#selection`

GitHub Copilot

Used 1 reference >

Here's a cleaned-up version of your code:

```
require 'daybreak'

class URLShortener
  def self.shorten(full_length_url, db)
    short_url = generate_short_url
    db.set!(short_url, full_length_url)
    short_url
  end

  def self.retrieve(short_url, db)
    raise "URL not found: #{short_url}" unless
      db[short_url]
  end

  private

  def self.generate_short_url(full_length_url, db)
    "https://conn.io/" +
      rand(1000..99999).to_s(36) +
      full_length_url
  end
end
```

Remove `db.close` calls

Remove unnecessary local variable

Use private method `generate_short_url`

+ Simpler design

~ (skip)

Here are the changes:

1. Extracted the short URL generation to a private method `generate_short_url` for better readability.
2. Removed the `db.close` calls. If the `shorten` and `retrieve` methods are called frequently, opening and closing the database connection each time could lead to a significant overhead. It's better to manage the database connection at a higher level of your application.
3. Simplified the `retrieve` method by removing the unnecessary `full_length_url` variable.

```
require 'daybreak'

class URLShortener
  def self.shorten(full_length_url, db)
    short_url = generate_short_url(full_length_url, db)
    db.set!(short_url, full_length_url)
    short_url
  end

  def self.retrieve(short_url, db)
    raise "URL not found: #{short_url}" unless
      db[short_url]
  end

  private

  def self.generate_short_url(full_length_url, db)
    "https://conn.io/" +
      rand(1000..99999).to_s(36) +
      full_length_url
  end
end
```

- Propose a cleanup to the code in `#selection`
- 🧠 I want to make incremental changes. Just do the change with the removal of the `db.close` methods for now

GitHub Copilot

Used 1 reference >

Sure, here's your code with just the `db.close` calls removed:

```
require 'daybreak'

class URLShortener
  def self.shorten(full_length_url, db)
    short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
    db.set!(short_url, full_length_url)
    short_url
  end

  def self.retrieve(short_url, db)
    unless db.keys.include?(short_url)
      raise "URL not found: #{short_url}"
    end

    full_length_url = db[short_url]
    full_length_url
  end
end
```

This change assumes that you're managing the database connection (opening and closing it) outside of these methods.

1

```
require 'daybreak'
```

```
class URLShortener
```

```
  def self.shorten(full_length_url, db)
```

```
    short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
```

```
    db.set!(short_url, full_length_url)
```

```
    short_url
```

```
  end
```

```
  def self.retrieve(short_url, db)
```

```
    raise "URL not found: #{short_url}" unless db.keys.include?(short_url)
```

```
    db[short_url]
```

```
  end
```

```
end
```

19

20

21

22





Suggest refactor of `#selection`, considering SOLID principles

GitHub Copilot

Used 1 reference >

The current code is quite simple and already adheres to many of the SOLID principles. However, there's one potential improvement that could be made in terms of the Single Responsibility Principle (SRP). Currently, the `URLShortener` class is responsible for both generating short URLs and interacting with the database. We could separate these responsibilities into two classes.

Here's a refactored version of your code:

```
require 'daybreak'

class URLShortener
  def initialize
    @db = Database.new
  end

  def shorten(full_length_url)
    short_url = generate_short_url
    @db.set!(short_url, full_length_url)
    short_url
  end

  def retrieve(short_url)
    raise "URL not found: #{short_url}" unless @db.keys.include?(short_url)

    @db[short_url]
  end

  private

  def generate_short_url
    "https://conn.io/" + rand(1000..99999).to_s(36)
  end
end

class Database
  def initialize
    @db = Daybreak::DB.new('data/urls.db')
  end

  def set!(key, value)
    @db.set!(key, value)
  end

  def get(key)
    @db[key]
  end

  def keys
    @db.keys
  end
end
```

Breaking change

In this version, `URLShortener` is responsible for generating short URLs, and `Database` is responsible for interacting with the database. This separation of concerns makes the code easier to understand and maintain. It also makes it easier to test each class in isolation.

```
1   require 'daybreak'
2
3   class URLShortener
4     def self.shorten(full_length_url, db)
5       short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
6       db.set!(short_url, full_length_url)
7       short_url
8     end
9
10    def self.retrieve(short_url, db)
11      raise "URL not found: #{short_url}" unless db.keys.include?(short_url)
12
13      db[short_url]
14    end
15  end
16
17
18
19
20
21
22
```

- Suggest refactor of `#selection`, considering SOLID principles
- We're initializing the DB connection outside of the class.
Can you propose a refactor that won't break the tests?

Used 1
The ci
impro
respoi



GitHub Copilot



reak'

Used 1 reference >

Sure, if you want to keep the database initialization outside of the class and not break the existing tests, you can refactor the code to separate the concerns of URL generation and database operations without changing how the database is initialized.

Here's a refactored version of your code:



Non-breaking change



Still not extensible

```
require 'daybreak'

class URLShortener
  attr_reader :db

  def initialize(db)
    @db = db
  end

  def shorten(full_length_url, db)
    short_url = URLGenerator.generate(full_length_url, db)
    db[short_url] = full_length_url
  end

  def retrieve(short_url, db)
    raise "URL not found: #{short_url}" unless db.keys.include?(short_url)


    db[short_url]
  end
end

class URLGenerator
  def self.generate(full_length_url, db)
    "https://conn.io/" + rand(1000..99999).to_s(36)
  end
end
```

```
def shorten(full_length_url, db)
  short_url = 'https://conn.io/' + rand(1000..99999).to_s(36)
  db[short_url] = full_length_url
end

def retrieve(short_url, db)
  raise "URL not found: #{short_url}" unless db.keys.include?(short_url)
  db[short_url]
end
```

In this version, `URLShortener` is responsible for interacting with the database, and `URLGenerator` is responsible for generating short URLs. This separation of concerns makes the code easier to understand and maintain. It also makes it easier to test each class in isolation.

- Suggest refactor of `#selection`, considering SOLID principles
- We're initializing the DB connection outside of the class. Can you propose a refactor that won't break the tests?
-  We want to easily swap out the URL generating logic. Could you suggest a refactor that would enable this?


Here's a refactored version of your

 GitHub Copilot

Used 1 reference >

Sure, if you want to make the URL generating logic swappable, you can introduce a strategy pattern. This involves defining a common interface for all URL generators and passing the specific generator you want to use as a parameter.

Here's a refactored version of your code:

 Non-breaking

 Still no

 Non-breaking change

 Now extensible

```
require "daybreak"

class URLShortener
  attr_reader :url_generator

  def initialize(url_generator = DefaultURLGenerator.new)
    @url_generator = url_generator
  end

  def shorten(full_length_url)
    short_url = URLGenerator.new(url_generator).generate(full_length_url)
    db.set!(short_url, full_length_url)
  end

  def retrieve(short_url)
    raise "URL not found: #{short_url}" unless db.keys.include?(short_url)

    db[short_url]
  end
end

class URLGenerator
  def generate(full_length_url)
    "https://conn.io/" + rand(1000..99999).to_s(36)
  end
end

class DefaultURLGenerator
  def generate(full_length_url)
    "https://conn.io/" + rand(1000..99999).to_s(36)
  end
end
```

```
https://conn.io/' + rand(1000..99999).to_s(36)
_url, full_length_url)
```

```
(short_url, db)
t found: #{short_url}" unless db.keys.include?(short_url)
```

In this version, `URLShortener.shorten` takes an optional `url_generator` parameter. If no generator is provided, it uses `DefaultURLGenerator`. This makes it easy to swap out the URL generating logic: just pass a different generator when you call `URLShortener.shorten`.

Each URL generator just needs to implement a `generate` method that returns a short URL. For example, here's how you could define a generator that always returns the same URL:

```
class StaticURLGenerator
  attr_reader :url

  def initialize(url)
    @url = url
  end

  def generate
    url
  end
end
```

 Uncovered edge case

And here's how you could use it:

```
URLShortener.shorten(full_length_url, db, StaticURLGenerator.new)
```

In this version, `URLShortener` is responsible for interacting with the database, and `URLGenerator` is responsible for generating short URLs. This separation of concerns makes the code easier to understand and maintain. It also makes it easier to test each class in isolation.

*Completing refactor
before addressing
newly-discovered
edge case...*

```
1   require 'daybreak'
2
3   class URLShortener
4     def self.shorten(full_length_url, db, url_generator = DefaultURLGenerator.new)
5       short_url = url_generator.generate
6       db.set!(short_url, full_length_url)
7       short_url
8     end
9
10    def self.retrieve(short_url, db)
11      raise "URL not found: #{short_url}" unless db.keys.include?(short_url)
12
13      db[short_url]
14    end
15  end
16
17  class DefaultURLGenerator
18    def generate
19      'https://conn.io/' + rand(1000..99999).to_s(36)
20    end
21  end
22 end
```





We don't want a static URL generator because all the short URLs would collide with each other and full URLs would be lost in the DB.



We don't want a static URL generator because all the short URLs would collide with each other and full URLs would be lost in the DB.

We don't handle the edge case that a generated short URL already exists in the DB.



We don't want a static URL generator because all the short URLs would collide with each other and full URLs would be lost in the DB.

We don't handle the edge case that a generated short URL already exists in the DB.

Can we add a test for that?

GitHub Copilot

Used 1 reference >

Sure, here's a simple unit test that checks if the `shorten` method can handle the case where a generated short URL already exists in the database. This test uses the `rspec` testing framework:

```
require 'rspec'
require_relative 'url_shortener'

describe URLShortener do
  it 'handles a short URL already in the database' do
    url_generator = double('URLGenerator').and_return('https://conn.io/1')
    db = double('Database').and_return(['https://conn.io/1'])

    expect(URLShortener.shorten('https://example.com', db, url_generator)).to eq('https://conn.io/2')
  end
end
```

This test uses a mock database and a mock URL generator. The URL generator is set up to return 'https://conn.io/1', the first URL generated by the URL generator. The database is set up to return 'https://conn.io/1', the first URL generated by the URL generator. The test then calls `URLShortener.shorten` and checks that it returns 'https://conn.io/2', the second URL generated by the URL generator. This indicates that the `shorten` method correctly handled the case where the first generated URL was already in the database.

Note that this test will fail with the current implementation of `URLShortener.shorten`, because it doesn't handle this case. To make the test pass, you would need to modify `URLShortener.shorten` to keep generating new URLs until it finds one that's not already in the database.

✓ Appropriate testing approach

✓ Well-explained

Can we add a test for that?



That works. We now have a failing test in `#terminalSelection`.

Can you now suggest an update to the implementation in `#selection`?

1
2
3

GitHub Copilot

Used 2 references >

Sure, to handle the case where a generated short URL already exists in the database, you can modify the `shorten` method to keep generating new URLs until it finds one that's not already in the database. Here's how you could do it:

```
def self.shorten(full_length_url, db, url_generator = DefaultURLGenerator.new)
  short_url = url_generator.generate
  while db.keys.include?(short_url)
    short_url = url_generator.generate
  end
  db.set!(short_url, full_length_url)
  short_url
end
```

In this version, after generating a short URL, the method checks if it's already in the database. If it is, it generates a new URL and checks again. This process repeats until it finds a URL that's not in the database. Then it stores the full URL under the short URL in the database and returns the short URL.

```
class URLShortener
  def self.shorten(full_length_url, db, url_generator = DefaultURLGenerator.new)
    short_url = url_generator.generate
    db.set!(short_url, full_length_url)
    short_url
  end

  def self.retrieve(short_url, db)
    raise "URL not found: #{short_url}" unless db.keys.include?(short_url)

    db[short_url]
  end
end
```

url to the db
hort URL if the generated one already exists in the database (FAILED - 1)
ngth url from the db
nd" error when url is not in the db

```
1) URLShortener.shorten generates a new short URL if the generated one already exists in the database
Failure/Error: expect(URLShortener.shorten(long_url, db, url_generator)).to eq(shorter_url)

expected: "https://conn.io/2"
got: "https://conn.io/1"

(compared using ==)
# ./spec/url_shortener_spec.rb:35:in `block (3 levels) in <top (required)>!'

Finished in 0.04193 seconds (files took 0.13735 seconds to load)
5 examples, 1 failure

Failed examples:

rspec ./spec/url_shortener_spec.rb:31 # URLShortener.shorten generates a new short URL if the generated one already exists in the database

zsh returned exit code 1
```

```

1 require './app/url_shortener'
2 require './spec_helper'
3
4 describe URLShortener do
5   let(:db) { double('Database') }
6   let(:url_generator) { double('URLGenerator') }
7
8   before do
9     allow(db).to receive(:set!)
10    allow(db).to receive(:keys).and_return([])
11  end
12
13  it 'generates a new short URL if the generated one already exists in the database' do
14    allow(url_generator).to receive(:generate).and_return('https://conn.io/1', 'https://conn.io/2')
15
16    allow(db).to receive(:keys).and_return(['https://conn.io/1'])
17
18    expect(URLShortener.shorten('https://example.com', db, url_generator)).to eq('https://conn.
19  end
20 end
21
22 RSpec.describe URLShortener do
23   before(:each) do
24     @db = Daybreak::DB.new './data/urls.db'
25   end
26
27   describe '.shorten' do
28     it 'shortens the url' do
29       short_url = URLShortener.shorten('https://www.yahoo.com', @db)
30
31       expect(short_url.size).to be < 'https://www.yahoo.com'.size
32     end
33   end
34 end

```

```

1 require 'daybreak'
2
3 class URLShortener
4   def self.shorten(full_length_url, db, url_generator = DefaultURLGenerator.new)
5     short_url = url_generator.generate
6     while db.keys.include?(short_url)
7       short_url = url_generator.generate
8     end
9     db.set!(short_url, full_length_url)
10    short_url
11  end
12
13  def self.retrieve(short_url, db)
14    raise "URL not found: #{short_url}" unless db.keys.include?(short_url)
15  end
16 end

```

```

URLShortener
  .shorten
    shortens the url
    writes shortened url to the db
    generates a new short URL if the generated one already exists in the database (FAILED - 1)
  .retrieve
    retrieves full-length url from the db
    raises a "not found" error when url is not in the db

```

Failures:

```

1) URLShortener.shorten generates a new short URL if the generated one already exists in the database
   Failure/Error: expect(URLShortener.shorten(long_url, db, url_generator)).to eq(shorter_url)
     expected: "https://conn.io/2"
     got: "https://conn.io/1"

     (compared using ==)
     # ./spec/url_shortener_spec.rb:35:in `block (3 levels) in <top (required)>!'

```

```

Finished in 0.04193 seconds (files took 0.13735 seconds to load)
5 examples, 1 failure

```

Failed examples:

```

rspec ./spec/url_shortener_spec.rb:31 # URLShortener.shorten generates a new short URL if the generated one already exists in the database

```

```

zsh returned exit code 1

```



Demo Review

Did GitHub Copilot help us improve Code Habitability?

+ Improved test speed and maintainability

- Memoized helper functions (`let`)
- Use of test doubles and stubs

+ Improved code design

- **Simplified behaviour:** removed unnecessary DB calls
- **Added extensibility:** dependency-injected logic for generating URLs

+ Miscellaneous improvements

- Helped identify an important edge case
- Expressive naming; readable code

- Not all suggestions provided were relevant

- Human coder(s) will always have more context than the AI Coding Assistant

- Needed repeated requests to make small, incremental changes

- Prompt customization techniques (tool-dependent) could improve this

- Suggestions lacked novelty

- What about using a constructor?

Guidelines

Guidelines

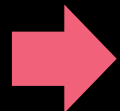
what to avoid → what to do instead

Context Poisoning

Recommendations based on code we intend to move away from

Review Fatigue

Glossing over code changes and missing details



Context Selection

Use context selectors: open files, `@workspace`, `#selection`, `#terminalSelection`, etc.



"Baby" Steps

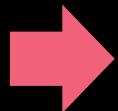
Ask the coding assistant to break down the steps

Guidelines

what to avoid → what to do instead

Context Poisoning

Recommendations based on code we intend to move away from



Context Selection

Use context selectors: open files, `@workspace`, `#selection`, `#terminalSelection`, etc.

Review Fatigue

Glossing over code changes and missing details



"Baby" Steps

Ask the coding assistant to break down the steps

Zoning Out on Autopilot

Insufficient critiquing of the emerging implementation



Interrogate Recursively

Repeatedly asking the coding assistant to improve on the code it just generated

Side-tracking

Getting distracted by surprising / useful coding assistant output



Pair Programming

Having a second human is still useful to help keep us on track

shipping code we can live with



shipping code that is a joy to live in

shipping code we can live with



psobocinski@thoughtworks.com
spikes.sobes.co



slides
repo
sources