



**OI-OPPA. European Social Fund
Prague & EU: We invest in your future.**

AE4B99RPH: Problem Solving and Games

Clean code.

Petr Pošík

Katedra kybernetiky
ČVUT FEL

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

The Sieve of

Eratosthenes: as a class?

Summary

Clean Code

Based on

**Robert C. Martin: *Clean Code: A Handbook of Agile Software Craftsmanship*,
Prentice Hall, 2008.**

Which of the following codes is cleaner? Why?

Two implementations of the same algorithm:

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value >= 2: # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]: # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else: # max_value < 2
        return list() # no primes, return empty list
```

```
PRIME = True
NONPRIME = False
```

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)

def init_integers_up_to(max_value):
    return [PRIME for i in range(max_value+1)]

def mark_non_primes(candidates):
    # Mark 0 and 1, they are not primes.
    candidates[0] = candidates[1] = NONPRIME
    for number in range(2, len(candidates)):
        if candidates[number] == PRIME:
            mark_as_not_prime_multiples_of(number, candidates)

def mark_as_not_prime_multiples_of(number, candidates):
    for multiple in range(2*number, len(candidates), number):
        candidates[multiple] = NONPRIME

def collect_remaining(candidates):
    primes = [i for i in range(len(candidates))
              if candidates[i]==PRIME]
    return primes
```

What is “clean code”?

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

The Sieve of

Eratosthenes: as a class?

Summary

Bjarne Stroustrup, author of C++ language and author of “The C++ Programming Language” book:

I like my code to be **elegant and efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease maintenance, error handling complete according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**

What is “clean code”?

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

The Sieve of

Eratosthenes: as a class?

Summary

Bjarne Stroustrup, author of C++ language and author of “The C++ Programming Language” book:

I like my code to be **elegant and efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease maintenance, error handling complete according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**

Grady Booch, author of “Object Oriented Analysis and Design with Applications” book:

Clean code is **simple and direct**. Clean code **reads like well-written prose**. Clean code **never obscures the designer’s intent** but rather is full of **crisp abstractions** and **straightforward lines of control**.

What is “clean code”?

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

The Sieve of

Eratosthenes: as a class?

Summary

Bjarne Stroustrup, author of C++ language and author of “The C++ Programming Language” book:

I like my code to be **elegant and efficient**. The logic should be **straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease maintenance, error handling complete according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**

Grady Booch, author of “Object Oriented Analysis and Design with Applications” book:

Clean code is **simple and direct**. Clean code **reads like well-written prose**. Clean code **never obscures the designer’s intent** but rather is full of **crisp abstractions** and **straightforward lines of control**.

Dave Thomas, OTI founder (acquired by IBM in 1996), Eclipse godfather:

Clean code can be read, and enhanced by a developer other than its **original author**. It has **unit and acceptance tests**. It has **meaningful names**. It provides one way rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and **provides a clear and minimal API**.

Clean code in practice

Clean Code

Which of the following codes is cleaner? Why?

What is "clean code"?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

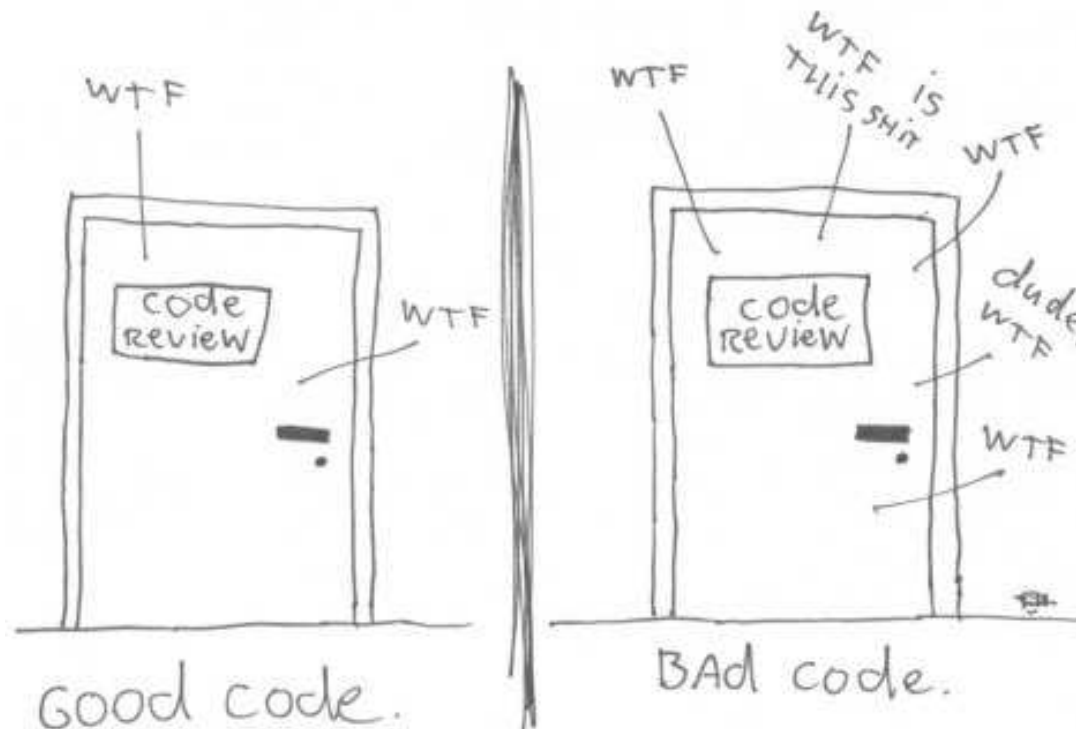
The Sieve of

Eratosthenes: as a class?

Summary

The only valid measurement of code quality: WTFs/minute

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>

Meaningful names

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of Eratosthenes: meaningful names

Comments
The Sieve of Eratosthenes: comments

Functions and methods
The Sieve of Eratosthenes: functions

The Sieve of Eratosthenes: as a class?

Summary

- ✓ It is **very hard** to come up with meaningful names! Put sufficient effort in it.
- ✓ Do not be afraid to change the name if you come up with better!
- ✓ Good name **reveals author’s intention**.
If a name requires a comment, it does not reveal its intention. Compare:
 - ✗ `self.d = 0` *# Elapsed time in days*
 - ✗ `self.elapsed_time_in_days = 0`
- ✓ Class names: **nouns** (with adjectives):
 - ✗ Customer, WikiPage, AddressParser, Filter, StupidFilter, Corpus, TrainingCorpus
- ✓ Function/method names: **verbs** (with objects):
 - ✗ post_payment, delete_page, save, train, test, get_email
- ✓ Single word for single concept! Do not use the same word for more than one purpose.
- ✓ Don’t be afraid of long names!
 - ✗ Long descriptive name is better than a long comment.
 - ✗ The larger the variable scope, the longer and more describing the variable name should be.
- ✓ Do not use magic numbers in the code! Use **named constants!**

The Sieve of Eratosthenes: meaningful names

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value >= 2: # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]: # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else: # max_value < 2
        return list() # no primes, return empty list
```

The Sieve of Eratosthenes: meaningful names

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value >= 2: # There are some primes
        # Initialize the list (incl. 0)
        f = [True for i in range(max_value+1)]
        # Get rid of the known non-primes
        f[0] = f[1] = False
        # Run the sieve
        for i in range(2, len(f)):
            if f[i]: # i is still a candidate
                # mark its multiples as not prime
                for j in range(2*i, len(f), i):
                    f[j] = False
        # Find the primes and put them in a list
        primes = [i for i in range(len(f)) if f[i]]
        return primes
    else: # max_value < 2
        return list() # no primes, return empty list
```

```
PRIME = True
NONPRIME = False
```

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value >= 2: # There are some primes
        # Initialize the list (incl. 0)
        candidates = [
            PRIME for i in range(max_value+1)]
        # Get rid of the known non-primes
        candidates[0] = candidates[1] = NONPRIME
        # Run the sieve
        for number in range(2, len(candidates)):
            if candidates[number]==PRIME:
                # mark its multiples as not prime
                for multiple in \
                    range(2*number, len(candidates), number):
                    candidates[multiple] = NONPRIME
        # Find the primes and put them in a list
        primes = [i for i in range(len(candidates))
                    if candidates[i]==PRIME]
        return primes
    else: # max_value < 2
        return list() # no primes, return empty list
```

Other meaningful names ahead!!!

Comments

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

The Sieve of

Eratosthenes: as a class?

Summary

Clean code (almost) does not need comments!

- ✓ Comments compensate for our failure to express ourselves in the programming language. Compare:

```
# Check to see if the employee is eligible for full benefits  
if (employee.flags & HOURLY_FLAG) and (employee.age > 65):
```

versus

```
if employee.is_eligible_for_full_benefits():
```

- ✓ Comments lie! Not always, not intentionally, but too often.
- ✓ Inaccurate comments are worse than no comments!
- ✓ Comments cannot repair bad code.
- ✓ Good comments:
 - ✗ little explanation, little clarification
 - ✗ emphasis, warning against consequences
 - ✗ TODOs
- ✓ Bad comments:
 - ✗ old (invalid), unimportant, unsuitable, redundant, or misleading comments
 - ✗ comments “because you have to comment”
 - ✗ commented-out code
 - ✗ non-local or irrelevant information

The Sieve of Eratosthenes: comments

```
# This function generates prime numbers up to  
# a user specified maximum. The algorithm  
# used is the Sieve of Eratosthenes.  
#  
# Eratosthenes of Cyrene, b. c. 276 BC,  
# Cyrene, Libya -- d. c. 194 BC, Alexandria.  
# The first man to calculate the circumference  
# of the Earth. Also known for working on  
# calendars with leap years and ran  
# the library at Alexandria.  
#  
# The algorithm is quite simple.  
# Given an array of integers starting at 2,  
# cross out all multiples of 2.  
# Find the next uncrossed integer,  
# and cross out all of its multiples.  
# Repeat until you have passed  
# the maximum value.  
#  
# @author hugo  
# @version 1
```

The Sieve of Eratosthenes: comments

```
# This function generates prime numbers up to  
# a user specified maximum. The algorithm  
# used is the Sieve of Eratosthenes.  
#  
# Eratosthenes of Cyrene, b. c. 276 BC,  
# Cyrene, Libya -- d. c. 194 BC, Alexandria.  
# The first man to calculate the circumference  
# of the Earth. Also known for working on  
# calendars with leap years and ran  
# the library at Alexandria.  
#  
# The algorithm is quite simple.  
# Given an array of integers starting at 2,  
# cross out all multiples of 2.  
# Find the next uncrossed integer,  
# and cross out all of its multiples.  
# Repeat until you have passed  
# the maximum value.  
#  
# @author hugo  
# @version 1
```

```
# This function generates prime numbers up to  
# a user specified maximum. The algorithm  
# used is the Sieve of Eratosthenes.  
# Given an array of integers starting at 2,  
# cross out all multiples of 2.  
# Find the next uncrossed integer,  
# and cross out all of its multiples.  
# Repeat until you have passed  
# the maximum value.  
#  
# @author hugo  
# @version 1
```

We will get rid of other comments in a while!

Functions and methods

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

The Sieve of

Eratosthenes: as a class?

Summary

- ✓ Functions shall be short! (And even shorter!)
- ✓ Function shall do a single thing and do it well. (And without side effects.)
- ✓ Ideally, functions shall be shorter than 5 lines. In that case:
 - ✗ they usually do exactly 1 thing.
 - ✗ they can have precise and meaningful name.
 - ✗ they cannot contain nested **if**, **for**, ... commands.
 - ✗ the blocks inside **if**, **for**, ... commands can be only a single line long.
- ✓ Short functions allow for testing individual parts of the program!
- ✓ Sections inside functions/methods:
 - ✗ A clear indication that the function/method does not do a single thing, and should be split up.
- ✓ Function/method parameters:
 - ✗ Keep their number small! 0, 1, 2, exceptionally 3.
 - ✗ Create the function/method name so that it evokes the order of arguments.
 - ✗ Boolean parameters usually suggest that the function/method does not do a single thing. Split it up!

The Sieve of Eratosthenes: functions

```
PRIME = True
NONPRIME = False
```

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value >= 2: # There are some primes
        # Initialize the list (incl. 0)
        candidates = [
            PRIME for i in range(max_value+1)]
        # Get rid of the known non-primes
        candidates[0] = candidates[1] = NONPRIME
        # Run the sieve
        for number in range(2, len(candidates)):
            if candidates[number]==PRIME:
                # mark its multiples as not prime
                for multiple in \
                    range(2*number, len(candidates), number):
                    candidates[multiple] = NONPRIME
        # Find the primes and put them in a list
        primes = [i for i in range(len(candidates))
                    if candidates[i]==PRIME]
        return primes
    else: # max_value < 2
        return list() # no primes, return empty list
```


The Sieve of Eratosthenes: functions

```
PRIME = True
NONPRIME = False
```

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value >= 2: # There are some primes
        # Initialize the list (incl. 0)
        candidates = [
            PRIME for i in range(max_value+1)]
        # Get rid of the known non-primes
        candidates[0] = candidates[1] = NONPRIME
        # Run the sieve
        for number in range(2, len(candidates)):
            if candidates[number]==PRIME:
                # mark its multiples as not prime
                for multiple in \
                    range(2*number, len(candidates), number):
                    candidates[multiple] = NONPRIME
        # Find the primes and put them in a list
        primes = [i for i in range(len(candidates))
                  if candidates[i]==PRIME]
        return primes
    else: # max_value < 2
        return list() # no primes, return empty list
```

```
PRIME = True
NONPRIME = False
```

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)

def init_integers_up_to(max_value):
    return [PRIME for i in range(max_value+1)]

def mark_non_primes(candidates):
    # Mark 0 and 1, they are not primes.
    candidates[0] = candidates[1] = NONPRIME
    for number in range(2, len(candidates)):
        if candidates[number] == PRIME:
            mark_as_not_prime_multiples_of(number, candidates)

def mark_as_not_prime_multiples_of(number, candidates):
    for multiple in range(2*number, len(candidates), number):
        candidates[multiple] = NONPRIME

def collect_remaining(candidates):
    primes = [i for i in range(len(candidates))
              if candidates[i]==PRIME]
    return primes
```

The Sieve of Eratosthenes: as a class?

```
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)

def init_integers_up_to(max_value):
    return [PRIME for i in range(max_value+1)]

def mark_non_primes(candidates):
    # Mark 0 and 1, they are not primes.
    candidates[0] = candidates[1] = NONPRIME
    for number in range(2, len(candidates)):
        if candidates[number] == PRIME:
            mark_as_not_prime_multiples_of(number, candidates)

def mark_as_not_prime_multiples_of(number, candidates):
    for multiple in range(2*number, len(candidates), number):
        candidates[multiple] = NONPRIME

def collect_remaining(candidates):
    primes = [i for i in range(len(candidates))
              if candidates[i]==PRIME]
```

The Sieve of Eratosthenes: as a class?

```
PRIME = True
NONPRIME = False
```

```
def generate_primes_up_to(max_value):
    """Find primes up to the max_value
    using the Sieve of Eratosthenes.

    """
    if max_value < 2:
        return []
    else:
        candidates = init_integers_up_to(max_value)
        mark_non_primes(candidates)
        return collect_remaining(candidates)

def init_integers_up_to(max_value):
    return [PRIME for i in range(max_value+1)]

def mark_non_primes(candidates):
    # Mark 0 and 1, they are not primes.
    candidates[0] = candidates[1] = NONPRIME
    for number in range(2, len(candidates)):
        if candidates[number] == PRIME:
            mark_as_not_prime_multiples_of(number, candidates)

def mark_as_not_prime_multiples_of(number, candidates):
    for multiple in range(2*number, len(candidates), number):
        candidates[multiple] = NONPRIME

def collect_remaining(candidates):
    primes = [i for i in range(len(candidates))
              if candidates[i]==PRIME]
```

```
PRIME = True
NONPRIME = False
```

```
class PrimesGenerator:
    """Prime numbers generator."""
    def __init__(self):
        self.candidates = []
        self.max = None

    def get_primes_up_to(self, max_value):
        """Return list of primes up to the max_value."""
        if max_value < 2: return []
        self.max = max_value+1
        self.init_candidates_up_to_max_value()
        self.mark_non_prime_candidates()
        return self.collect_remaining_candidates()

    def init_candidates_up_to_max_value(self):
        self.candidates = [PRIME for i in range(self.max)]

    def mark_non_prime_candidates(self):
        # Cross out 0 and 1, they are not primes.
        self.candidates[0] = self.candidates[1] = NONPRIME
        for number in range(2, int(self.max**0.5)+1):
            if self.candidates[number]==PRIME:
                self.mark_as_not_prime_multiples_of(number)

    def mark_as_not_prime_multiples_of(self, number):
        for multiple in range(2*number, self.max, number):
            self.candidates[multiple] = NONPRIME

    def collect_remaining_candidates(self):
        return [i for i in range(self.max)
              if self.candidates[i]==PRIME]
```

Summary

Clean Code

Which of the following codes is cleaner? Why?

What is “clean code”?

Clean code in practice

Meaningful names

The Sieve of

Eratosthenes:

meaningful names

Comments

The Sieve of

Eratosthenes: comments

Functions and methods

The Sieve of

Eratosthenes: functions

The Sieve of

Eratosthenes: as a class?

Summary

- ✓ Clean code is a subjective concept, yet:
 - ✗ there are some generally accepted features of clean code, and
 - ✗ all programmers shall strive for it.
- ✓ Clean code shall be foremost readable (almost like sentences in natural language).
- ✓ 80 % of clean code are well chosen names!
- ✓ Suitable names can be chosen if the functions are short!
- ✓ If your program contains repeated pieces of almost the same code, it is almost always possible to define it as a new function/method.



**OI-OPPA. European Social Fund
Prague & EU: We invest in your future.**
