A4B99RPH: Řešení problémů a hry
Čistý kód.

Petr Pošík

Katedra kybernetiky
ČVUT FEL

Clean Code

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

Který kód je čistší? A proč?

Dvě implementace téhož algoritmu:

```
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
```


Co je “clean code”?

Bjarne Stroustrup, autor jazyka C++ a knihy “The C++ Programming Language”:

> 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, autor knihy “Object Oriented Analysis and Design with Applications”:

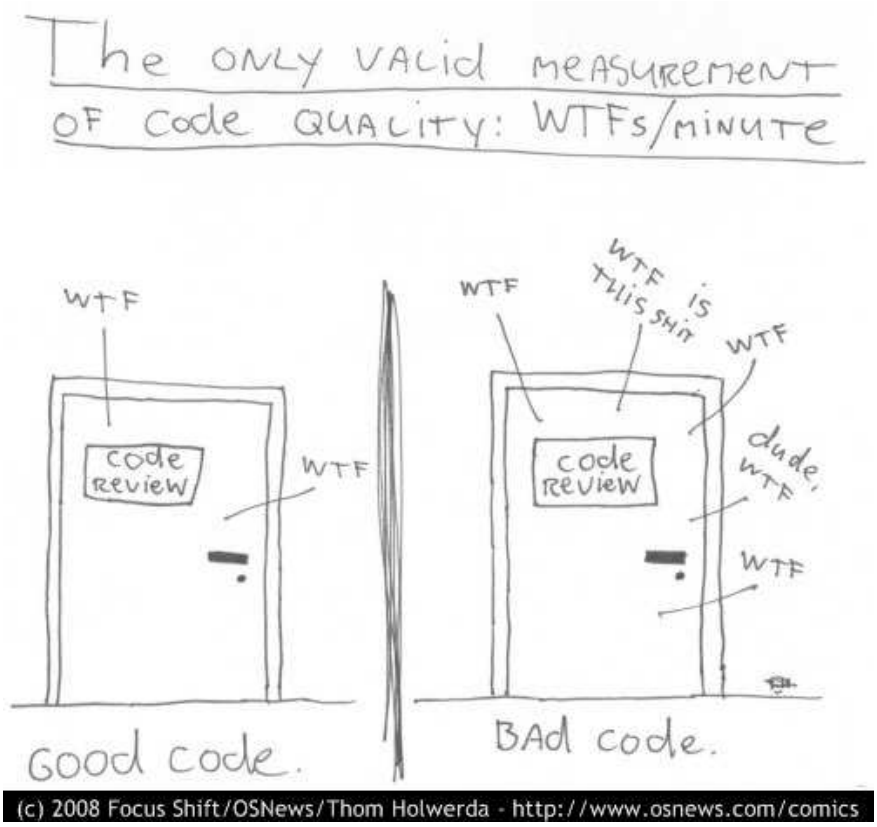
> Clean code is **simple and direct**. Clean code **reads like well-written prose**. Clean code **never obscures the designer’s intent** but rather is full of **crisp abstractions** and **straightforward lines of control**.

Dave Thomas, zakladatel firmy OTI (převzata firmou IBM v roce 1996), kmotr Eclipse:

> **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**.

Čistý kód v praxi

Jediné správné měřítko kvality kódu: Co-to-k-čerty za minutu



Smysluplná jména

- Vymyslet dobrá jména je **velmi těžké**! Věnujte tomu dostatečnou pozornost!
- Nebojte se jméno změnit, přijdete-li na lepší!
- Dobré jméno **odhaluje autorův záměr** (intention-revealing).
  Pokud jméno vyžaduje komentář, neodhaluje záměr. Porovnejte:
  - `self.d = 0  # Elapsed time in days`
  - `self.elapsed_time_in_days = 0`
- Názvy tříd: **podstatná jména** (s přívlastky):
  - `Customer`, `WikiPage`, `AddressParser`, `Filter`, `StupidFilter`, `Corpus`, `TrainingCorpus`
- Názvy funkcí/metod: **slovesa** (s předmětem):
  - `post_payment`, `delete_page`, `save`, `train`, `test`, `get_email`
- Jeden termín pro jeden koncept! Nepoužívejte stejné slovo k více účelům!
- Nebojte se dlouhých jmen!
  - Dlouhé popisné jméno je lepší než dlouhý popisný komentář.
  - Čím delší oblast platnosti proměnné, tím popisnější jméno by měla mít.
- Používejte **pojmenované konstanty** místo magických čísel v kódu!

Eratostenovo síto: smysluplná jména

```python
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
```

```python
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
```

Další smysluplná jména budou následovat!!!

Komentáře

Čistý kód komentáře (skoro) nepotřebuje!

- Komentáře kompenzují naše selhání vyjádřit se v prog. jazyce. Porovnej:

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

  versus

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

- Komentáře lžou! Ne vždy a ne záměrně, ale až příliš často!
- Nepřesné komentáře jsou horší než žádné komentáře!
- Komentáře nenapraví špatný kód!
- Dobré komentáře:
  - (do)vysvětlení, (do)upřesnění
  - zdůraznění, varování před následky
  - TODOs
- Špatné komentáře:
  - staré (už neplatné), bezvýznamné, nevhodné, redundantní, nebo zavádějící komentáře
  - komentáře z povinnosti
  - zakomentovaný kód
  - nelokální nebo nadbytečné informace

Eratostenovo síto: komentáře

```
# 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
```

Za chvíli se zbavíme dalších komentářů!

Funkce a metody

- Funkce by měly být krátké! (A ještě kratší!)
- Funkce by měla dělat právě 1 věc a měla by ji dělat dobře. (A bez vedlejších efektů.)
- Funkce dlouhé méně než 5 řádků:
  - Většinou dělají právě 1 věc.
  - Mohou mít přesné a výstižné jméno.
  - Nemohou obsahovat vnořené příkazy `if`, `for`, ...
  - Bloky uvnitř příkazů `if`, `for`, ... jsou pouze 1 řádek dlouhé
- Krátké funkce umožňují testovat dílčí části algoritmu!
- Sekce uvnitř funkcí/metod:
  - Jasná indikace toho, že funkce/metoda nedělá jen 1 věc a měla by být rozdělena.
- Argumenty funkcí/metod:
  - Udržujte jejich počet malý! 0, 1, 2, výjimečně 3.
  - Vytvořte jméno tak, aby evokovalo pořadí argumentů.
  - Boolovské argumenty funkcí často značí, že funkce nedělá 1 věc! Rozdělte ji.

Eratostenovo síto: funkce

```
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
```


Eratostenovo síto: převod na třídu

```
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]
```

Eratostenovo síto: funkce a třída?

```
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]
```

```
PRIME = True
NONPRIME = False

def generate_primes_up_to(max_value):
    """Return a list of primes up to the max_value."""
    if max_value < 2: return []
    candidates = CandidateNumberList(max_value)
    candidates.checkout_multiples()
    return candidates.collect_remaining()

class CandidateNumberList:
    """List of boolean values for use in the Sieve of Eratosthenes.
    Shall be used with the generate_primes_up_to function.
    """
    def __init__(self, max_value):
        self.max = max_value + 1
        self.candidates = [PRIME for i in range(self.max)]
        self.candidates[0] = self.candidates[1] = NONPRIME

    def checkout_multiples(self):
        """Mark multiples of all prime numbers as not prime."""
        for number in range(2, int(self.max**0.5)+1):
            if self.candidates[number] == PRIME:
                self.checkout_multiples_of(number)

    def checkout_multiples_of(self, number):
        """Mark multiples of number as not prime."""
        for multiple in range(2*number, self.max, number):
            self.candidates[multiple] = NONPRIME

    def collect_remaining(self):
        """Return a list of remaining candidates, they are prime."""
        return [i for i in range(self.max)
                    if self.candidates[i]==PRIME]
```


Závěr

- Čistý kód je subjektivní pojem, přesto by se o něj měl každý programátor snažit.
- Čistý kód by měl být především čitelný (skoro jako v přirozeném jazyce).
- 80 % čistého kódu jsou správně zvolená jména!
- Vhodná jména lze volit, jsou-li funkce/metody dostatečně krátké!
- Opakují-li se ve vašem programu stejné nebo podobné kusy kódu, prakticky vždy je možné takový kód definovat jako samostatnou funkci/metodu.