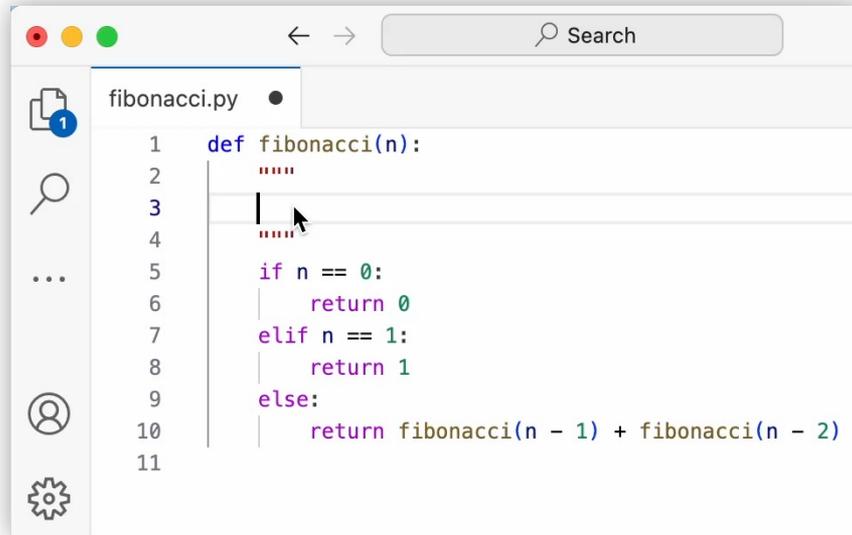


Can Developers Prompt? A Controlled Experiment for Code Documentation Generation

Hans-Alexander Kruse, [Tim Puhlfürß](#), Walid Maalej



Creating Source Code Documentation



The screenshot shows a code editor window titled 'fibonacci.py'. The code is as follows:

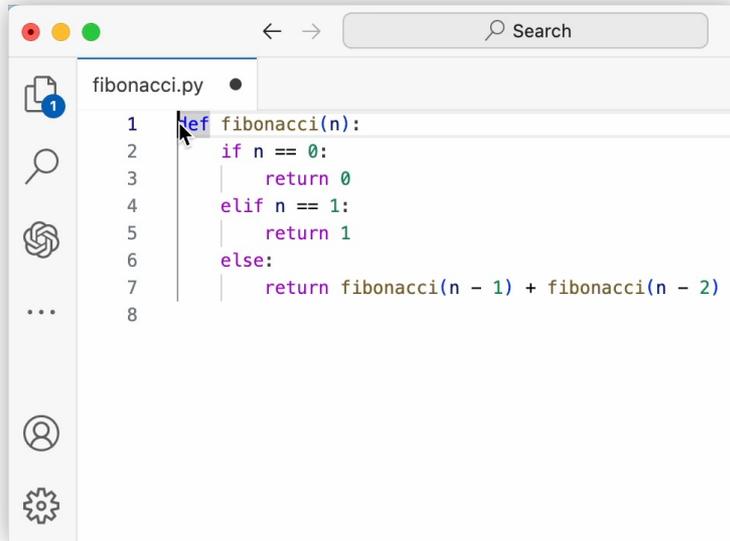
```
1 def fibonacci(n):
2     """
3
4     """
5     if n == 0:
6         return 0
7     elif n == 1:
8         return 1
9     else:
10        return fibonacci(n - 1) + fibonacci(n - 2)
11
```

The code is mostly empty, with only a few lines of docstring and logic. A mouse cursor is pointing at the end of the docstring on line 4.

- ⚡ Tedious process
- ⚡ Undetailed / too detailed documentation
- ⚡ Incomprehensible to others

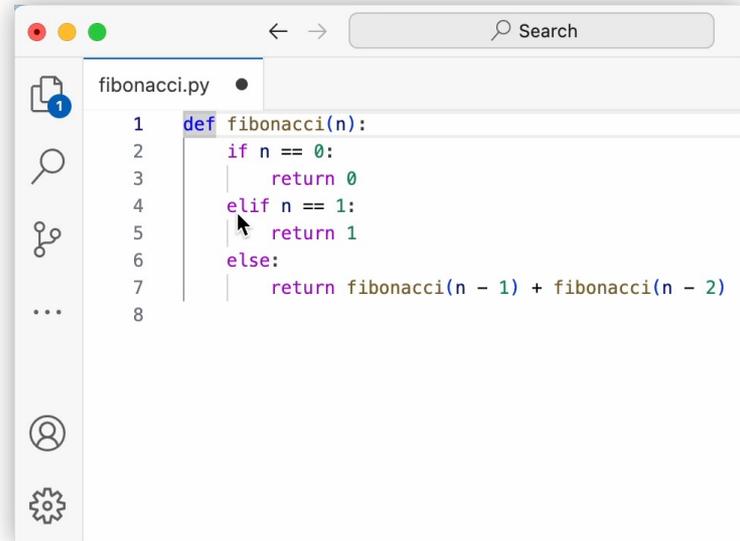
Generating Source Code Documentation

Ad-hoc prompt



```
1 def fibonacci(n):
2     if n == 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
7         return fibonacci(n - 1) + fibonacci(n - 2)
8
```

Predefined prompt

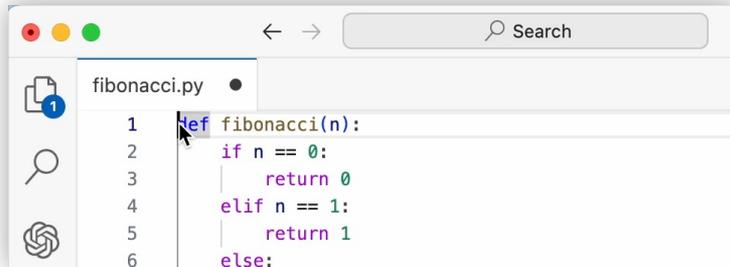


```
1 def fibonacci(n):
2     if n == 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
7         return fibonacci(n - 1) + fibonacci(n - 2)
8
```

Research Questions

- RQ1:** How well can developers prompt LLMs to generate code documentation compared to a predefined prompt?
- RQ2:** How is the developer experience for ad-hoc prompting compared to executing predefined prompts?

Ad-hoc prompt

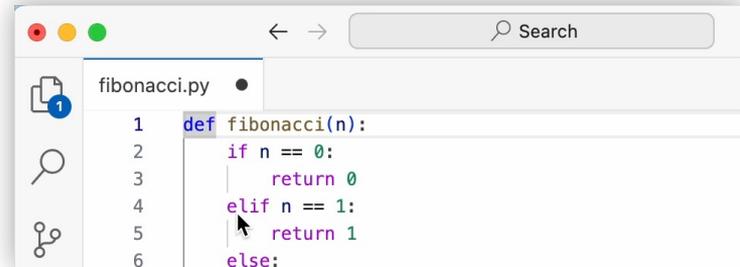


A screenshot of a code editor window titled 'fibonacci.py'. The code is as follows:

```
1 def fibonacci(n):
2     if n == 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
```

A mouse cursor is positioned over the 'def' keyword on line 1. The editor interface includes a search bar at the top and a sidebar on the left with icons for file management, search, and AI assistance.

Predefined prompt



A screenshot of a code editor window titled 'fibonacci.py'. The code is as follows:

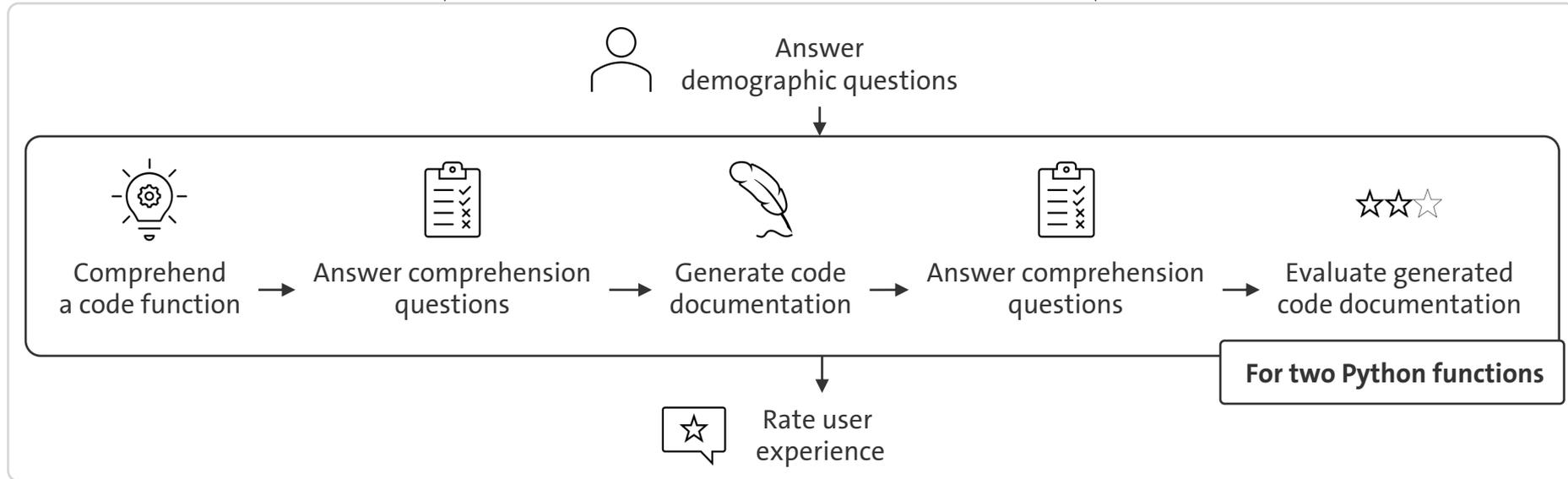
```
1 def fibonacci(n):
2     if n == 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
```

A mouse cursor is positioned over the 'return 1' statement on line 5. The editor interface is identical to the 'Ad-hoc prompt' screenshot, showing a search bar and sidebar.

Methodology

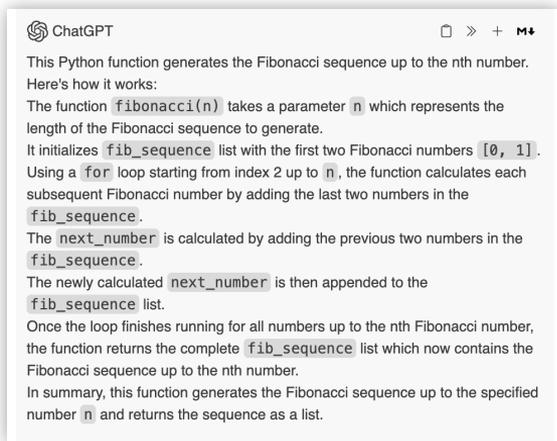
Experimental group (25 participants):
ad-hoc prompt

Control group (25 participants):
predefined prompt



Developers are not necessarily prompting experts

▶ ~50% failed the first attempt



ChatGPT

This Python function generates the Fibonacci sequence up to the nth number. Here's how it works:

The function `fibonacci(n)` takes a parameter `n` which represents the length of the Fibonacci sequence to generate.

It initializes `fib_sequence` list with the first two Fibonacci numbers `[0, 1]`.

Using a `for` loop starting from index 2 up to `n`, the function calculates each subsequent Fibonacci number by adding the last two numbers in the `fib_sequence`.

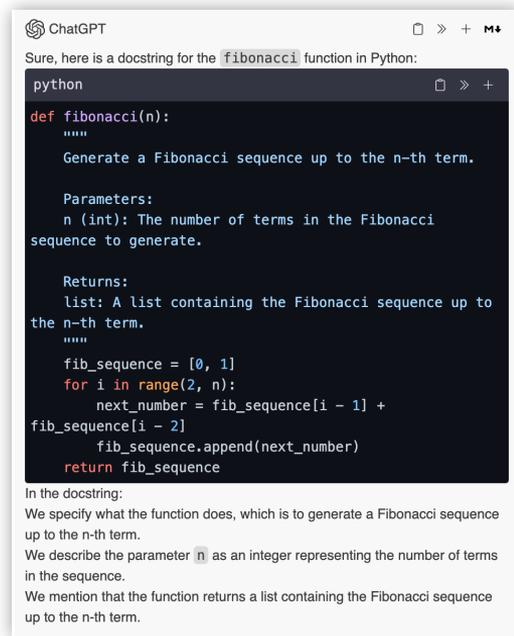
The `next_number` is calculated by adding the previous two numbers in the `fib_sequence`.

The newly calculated `next_number` is then appended to the `fib_sequence` list.

Once the loop finishes running for all numbers up to the nth Fibonacci number, the function returns the complete `fib_sequence` list which now contains the Fibonacci sequence up to the nth number.

In summary, this function generates the Fibonacci sequence up to the specified number `n` and returns the sequence as a list.

👉 “Explain this function”
vs.
“Write a Docstring” 👉



ChatGPT

Sure, here is a docstring for the `fibonacci` function in Python:

```
python
def fibonacci(n):
    """
    Generate a Fibonacci sequence up to the n-th term.

    Parameters:
    n (int): The number of terms in the Fibonacci
    sequence to generate.

    Returns:
    list: A list containing the Fibonacci sequence up to
    the n-th term.
    """
    fib_sequence = [0, 1]
    for i in range(2, n):
        next_number = fib_sequence[i - 1] +
        fib_sequence[i - 2]
        fib_sequence.append(next_number)
    return fib_sequence
```

In the docstring:

We specify what the function does, which is to generate a Fibonacci sequence up to the n-th term.

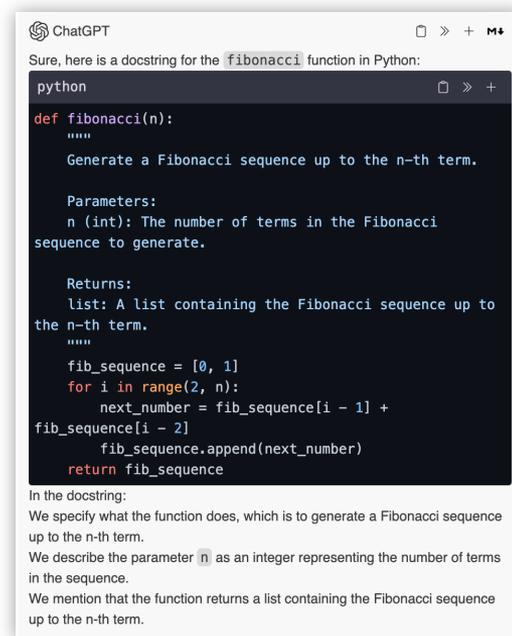
We describe the parameter `n` as an integer representing the number of terms in the sequence.

We mention that the function returns a list containing the Fibonacci sequence up to the n-th term.

Developers are not necessarily prompting experts

✓ Successful ad-hoc prompting attempts:

Prompt pattern	Students	Professionals
Write a comment	24	10
Write a Docstring	3	9
Explain the function in a comment	2	0
Write a Python-conform description	1	0
Summarize the function in a comment	0	1



ChatGPT

Sure, here is a docstring for the `fibonacci` function in Python:

```
python
def fibonacci(n):
    """
    Generate a Fibonacci sequence up to the n-th term.

    Parameters:
    n (int): The number of terms in the Fibonacci
    sequence to generate.

    Returns:
    list: A list containing the Fibonacci sequence up to
    the n-th term.
    """
    fib_sequence = [0, 1]
    for i in range(2, n):
        next_number = fib_sequence[i - 1] +
        fib_sequence[i - 2]
        fib_sequence.append(next_number)
    return fib_sequence
```

In the docstring:

- We specify what the function does, which is to generate a Fibonacci sequence up to the n-th term.
- We describe the parameter `n` as an integer representing the number of terms in the sequence.
- We mention that the function returns a list containing the Fibonacci sequence up to the n-th term.

Evaluating code documentation quality requires human-centered quality metrics

Group	Subgroup	Code Function	Grammatical Correctness	Readability	Missing Information	Unnecessary Information	Usefulness	Helpfulness
Ad-hoc	Students	1	4.86 (0.35)	4.0 (1.0)	4.6 (0.63)	3.33 (1.35)	3.93 (0.88)	4.46 (0.64)
Predefined	Students	1	4.93 (0.26)	4.66 (0.62)	4.6 (0.83)	4.86 (0.35)	4.6 (0.63)	4.6 (0.63)
p-val. δ	Students	1	0.58 0.22	0.03 0.8	0.76 0.0	0.0002 1.56	0.03 0.87	0.52 0.21
Ad-hoc	Students	2	4.4 (1.06)	3.2 (1.47)	3.86 (1.30)	2.66 (1.68)	3.13 (1.41)	3.66 (1.23)
Predefined	Students	2	4.73 (0.80)	4.6 (0.51)	4.06 (0.96)	4.86 (0.35)	4.73 (0.46)	4.6 (0.74)
p-val. δ	Students	2	0.13 0.36	0.002 1.27	0.84 0.17	0.0002 1.82	0.0008 1.53	0.02 0.92
Ad-hoc	Professionals	1	4.9 (0.32)	4.3 (0.67)	4.7 (0.95)	3.2 (1.03)	4.4 (0.70)	4.6 (0.52)
Predefined	Professionals	1	4.8 (0.42)	4.8 (0.42)	4.0 (1.41)	3.5 (1.65)	4.1 (1.45)	3.9 (1.45)
p-val. δ	Professionals	1	0.58 0.27	0.07 0.89	0.08 0.58	0.51 0.22	0.97 0.26	0.38 0.64
Ad-hoc	Professionals	2	4.6 (0.70)	4.0 (1.15)	4.0 (1.33)	3.1 (1.52)	4.1 (1.20)	4.5 (0.53)
Predefined	Professionals	2	4.8 (0.42)	4.8 (0.42)	4.3 (0.67)	4.4 (1.26)	4.4 (1.07)	4.6 (0.97)
p-val. δ	Professionals	2	0.58 0.35	0.06 0.92	0.9 0.28	0.07 0.93	0.47 0.26	0.28 0.13

Students significantly preferred results of predefined prompts.

-  Readability
-  Unnecessary information
-  Usefulness
-  Helpfulness

No statistically significant differences among professionals

Beneficial multidimensional, semantic metric

Automated execution via LLMs possible?

Generating documentation with LLMs is an iterative task

Prompting preferences of participants:

- Students: predefined (→ **guidance**)
- Professionals: predefined + ad-hoc (→ **flexibility**)

Outcome preferences:

- Documentation generated by **predefined prompt**

Nevertheless:

Generated documentation often perceived as basis for further refinement

Summary

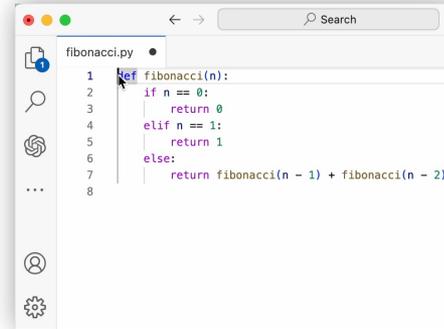
Can developers prompt?

- Not necessarily.
- Generating documentation with LLMs requires:
 - Prompting skills
 - Multidimensional, semantic metrics
 - Iterations

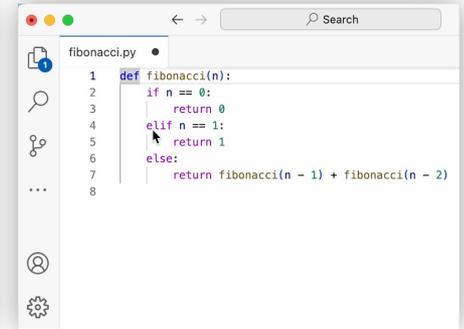
Contact:

tim.puhlfuerss@uni-hamburg.de

walid.maalej@uni-hamburg.de



```
1 def fibonacci(n):
2     if n == 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
7         return fibonacci(n - 1) + fibonacci(n - 2)
8
```



```
1 def fibonacci(n):
2     if n == 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
7         return fibonacci(n - 1) + fibonacci(n - 2)
8
```

Paper preprint and further links →

