Koding dan Kopi

Pengenalan

Bismillah.

So kat sini abam nak share sedikit artikel yang berkaitan dengan programming/software development/software engineering. Kalau ada kesempatan, kita akan buat secara bersiri.

So are you guys ready to embark on a coding journey? Sama ada korang otai dalam programming atau baru setahun jagung, Insyallah anda akan dapat sedikit sebanyak ilmu atau sebagai refresher dalam siri artikel yang abam ingin kongsi. So, grab your 3-in-1 stick, pour some hot water, sit back, and let's dive into the wonderful world of programming!

Ok untuk siri pengenalan ni abam ingin mulakan dengan Cognitive Complexity.

Apa itu Cognitive Complexity?

Cognitive Complexity refers to the ***mental effort required to understand a particular piece of code or a software system.***

Atau dalam bahasa mudahnya : betapa susahnya untuk memahami sesuatu kod.

Cognitive complexity is an important aspect of software development because complex code can be difficult to maintain, understand, and modify. In fact, cognitive complexity is often used as a metric to measure the quality of software code.

Cuba ambil contoh kod di bawah:

```
function freqWords(t, n) {
  var map = {}, word, words = t.toLowerCase().split(/\W+/), list = [];

  for (var i = 0; i < words.length; i++) {
    word = words[i];
    if (map[word]) {
      map[word]++;
    } else {
      map[word] = 1;
    }
  }

  for (word in map) {
    if (map.hasOwnProperty(word)) {
      list.push([word, map[word]]);
    }
  }

  list.sort(function(a, b) {
    return b[1] - a[1];
  });

  var topN = [];
  for (var j = 0; j < Math.min(n, list.length); j++) {
    topN.push(list[j][0]);
  }

  return topN;
}
```

(CHUP!! Kita guna full english la eh? Macam pelik pulak bunyi...)

The above code is difficult to read and understand due to the nested conditional statements, and it becomes more challenging to modify and maintain it over time.

While this is an example of how we can improve the cognitive complexity of the previous code:

```
function findMostFrequentWords(text, numWords) {
  const wordMap = new Map();
  const words = text.toLowerCase().split(/\W+/);

  for (let word of words) {
    if (wordMap.has(word)) {
      wordMap.set(word, wordMap.get(word) + 1);
    } else {
      wordMap.set(word, 1);
    }
  }

  const sortedWords = Array.from(wordMap.entries())
    .sort((a, b) => b[1] - a[1])
    .map(entry => entry[0])
    .slice(0, numWords);

  return sortedWords;
}
```

In this version, we've taken a parang to our code and chopped it into bite-sized functions that even Opah pada Upin Ipin could understand. We also ditched the endless nesting of conditional statements and replaced them with fancy maps and objects that store our data like it's their job (which, let's face it, it is). Now our code is as sleek and streamlined as a cheetah on roller skates, and we're feeling pretty good about it.

The code is now easier to read and modify, making it more maintainable and less prone to errors.

Improving your code

So, how do we improve Cognitive Complexity in our code? Here are some guides :

1. Don't write functions that are bigger than a T-Rex. Break them down into smaller, more manageable pieces like a bag of Skittles.
2. Don't name your variables like you're playing Sahibba. Use descriptive names that are easy to understand, like naming your cat "SiGebus" instead of "Cgbus."
3. Don't nest your loops like a Russian doll (If you don’t know what they are, pause and quickly google them ☺). Try using helper functions or dictionaries to make things simpler, like finding your keys in a bowl of Honey Stars.
4. Don't reinvent the wheel, use the built-in features of your programming language like a pro. It's like using a cheat code in a video game, but totally legal.
5. Don't repeat yourself like a broken record. Write reusable functions that you can use over and over again, like a favourite pair of socks.
6. Don't write comments that sound like you're writing a novel. Keep them short and sweet like a tweet, and they'll be much easier to read.

Conclusion

In conclusion, cognitive complexity is a critical concept for all developers to understand. By designing code that is easy to read and understand, we can reduce the chances of introducing bugs, improve maintainability, and make it easier for other developers to work with our code. We've discussed some practical tips for improving cognitive complexity, including breaking down code into smaller functions, avoiding nested loops, and using descriptive variable names. Remember, writing code that is easy to understand doesn't mean sacrificing performance or functionality. With these tips, you can write code that is both elegant and easy to understand, seperti kata-kata Tok Dalang; “Ditelan tak sengkang...”. So go forth, fellow developers, and write code that would make even Usman Awang proud! (Err.. Do you guys know him? Haha)

Done reading? Good job! Here’s a joke for your trophy

Artikel ini disediakan oleh:

Shah Erhan
Penyair Koding