Frågor till intervju om bra/”clean” kod

Start

Kort inledning om struktur, syfte och demografiska frågor. Förklara att anteckningar kommer tas under intervjun.

Syfte/Mål: Upprätthålla vetenskaplighet, replikerhet, underlätta vid utvärdering av eventuell partiskhet (bias).

1. Vilken team-roll har du?
2. Antal års erfarenhet av mjukvaruutveckling? (Fråga specifikt om C++)
3. Tidigare erfarenhet i att göra kodgranskningar, exempelvis genom code reviews?

Kodsnuttar

I denna den av kommer vi att titta på ett fåtal kodsnuttar och diskutera hur enkla eller svåra de är att förstå och arbeta med.

Syfte/Mål: Kommentarer om kodsnuttarnas utformning, vad som är bra och dåligt samt vilka karaktärsdrag som bidrar till detta. Frågorna är utformade med en specifik utgångspunkt men är tänkt att öppna för diskussioner som relaterar till ”clean code” och hur bra programkod ska vara utformad. (open-ended/semi-structured)

1. Vad är det första du tittar på när du försöker förstå koden? (Täck in läsbarhet och struktur)
2. Hade du varit bekväm att arbeta med den här koden? Varför?(Täck in underhållbarhet och flexibilitet.)
3. Hur testbar upplever du denna kod? Vad gör att kod är testbar för dig? (Täck in testbarhet)

- ”clean code”-läsbarhetsaspekter, t.ex. namnsättning, nästlade strukturer, radlängd, storlek, kommentarer, logiskt flöde (boolska uttryck), återupprepning.

Öppna frågor

I denna del diskuterar vi mer generellt vad som är bra kod, samt i vilka situationer olika karaktärsdrag på kod är viktigt utan att utgå från givna exempel.

Syfte/Mål: Få fram deltagarnas generella syn på vad som gör kod bra och ge möjlighet att få fram eventuella perspektiv som missades i kodsnuttarna.

1. Hur skulle du definera kod med hög kvalitet?
2. Upplever du att man måste kompromissa mellan olika kvalitetsaspekter (t.ex. läsbarhet vs. testbarhet vs. underhållbarhet)?
   (a) Om ja, på vilket sätt påverkar kontexten (t.ex. bygga ny modul vs. ändring av befintlig kod) hur du kompromissar med kvalitetaspekter?
3. Följer du någon specifik modell eller något tankesätt när du skriver kod för att säkerställa kvalitet? (t.ex. kort och konsist, följa en viss arkitektur/design, tydligt dokumenterad, ”clean code”, externt verktyg)
   (a) Har ni annat tankesätt för kod som skrivs för team/som ska reviewas kontra kod ni skriver privat/hemma? (Tillagd spontant under första intervjun.)
4. Vad i källkod ser du bidrar till att den blir svår att underhålla eller arbeta med över tid?
5. Har du några andra tankar kring vad bra programkod är?

Questions for an Interview on Good/"Clean" Code

Introduction

A brief explanation of the structure, purpose, and demographic questions. Clarify that notes will be taken during the interview.

Purpose/Goal: Maintain scientific rigor, reproducibility, and facilitate evaluation of potential bias.

1. What is your team role?
2. How many years of experience do you have in software development? (Ask specifically about C++)
3. Have you had any previous experience conducting code reviews (for example, through formal code review processes)?

Code Snippets

In this section, we will look at a few code snippets and discuss how easy or difficult they are to understand and work with.

Purpose/Goal: Collect comments on the structure of the snippets—what is good or bad—and which characteristics contribute to that. The questions are formed with a specific perspective but are meant to open up discussions related to "clean code" and how good program code should be designed. (open-ended/semi-structured)

1. What is the first thing you look at when trying to understand the code? (Covers readability and structure)
2. Would you feel comfortable working with this code? Why? (Covers maintainability and flexibility)
3. How testable do you find this code? What makes code testable for you? (Covers testability)

- "Clean code" readability aspects, e.g., naming conventions, nested structures, line length, file size, comments, logical flow (Boolean expressions), repetition.

Open-ended Questions

In this section, we discuss more generally what constitutes good code and in which situations various code characteristics are important, without relying on the given examples.

Purpose/Goal: Elicit the participants' general views on what makes code good and provide an opportunity to capture any perspectives that might have been missed in the code snippets.

1. How would you define high-quality code?
2. Do you find that you must compromise between different quality aspects (e.g., readability vs. testability vs. maintainability)?
   (a) If yes, in what way does the context (e.g., building a new module vs. modifying existing code) affect how you compromise on quality aspects?
3. Do you follow any specific model or mindset when writing code to ensure quality? (e.g., concise code, adhering to certain architecture/design principles, clear documentation, "clean code," external tools)
   (a) Do you adopt a different mindset for code that will be reviewed or used by a team versus code you write privately/at home? (Added spontaneously during the first interview.)
4. What do you see in source code that makes it difficult to maintain or work with over time?
5. Do you have any other thoughts on what constitutes good program code?

Exempel C++ Kodsnuttar

Slumpmässigt utplockade funktioner från några av de mest populära C++ open-source projekten på GitHub.

React Native

```cpp
bool Value::strictEquals(Runtime& runtime, const Value& a, const Value& b) {
  if (a.kind_ != b.kind_) {
    return false;
  }
  switch (a.kind_) {
    case UndefinedKind:
    case NullKind:
      return true;
    case BooleanKind:
      return a.data_.boolean == b.data_.boolean;
    case NumberKind:
      return a.data_.number == b.data_.number;
    case SymbolKind:
      return runtime.strictEquals(
          static_cast<const Symbol&>(a.data_.pointer),
          static_cast<const Symbol&>(b.data_.pointer));
    case BigIntKind:
      return runtime.strictEquals(
          static_cast<const BigInt&>(a.data_.pointer),
          static_cast<const BigInt&>(b.data_.pointer));
    case StringKind:
      return runtime.strictEquals(
          static_cast<const String&>(a.data_.pointer),
          static_cast<const String&>(b.data_.pointer));
    case ObjectKind:
      return runtime.strictEquals(
          static_cast<const Object&>(a.data_.pointer),
          static_cast<const Object&>(b.data_.pointer));
    }
  return false;
}
```

```
void Browser::Exit(gin::Arguments* args) {
  int code = 0;
  args->GetNext(&code);

  if (!ElectronBrowserMainParts::Get()->SetExitCode(code)) {
    // Message loop is not ready, quit directly.
    exit(code);
  } else {
    // Prepare to quit when all windows have been closed.
    is_quitting_ = true;

    // Remember this caller so that we don't emit unrelated events.
    is_exiting_ = true;

    // Must destroy windows before quitting, otherwise bad things can happen.
    if (electron::WindowList::IsEmpty()) {
      Shutdown();
    } else {
      // Unlike Quit(), we do not ask to close window, but destroy the window
      // without asking.
      electron::WindowList::DestroyAllWindows();
    }
  }
}
```