Clean Code, Horrible performance

SE RADIO EPISODE 577

Álvaro Ibáñez Coedo	UO282206
Sergio Quintana Fernández	UO288090
Jorge Joaquín Gancedo Fernández	UO282161
Miguel Estapé Fernández	UO277566

Table of contents

1. Introduction

This interview had as its guest Casey Muratori, who is a game developer working on different projects. During this interview the main topic discussed was how clean code can affect a program's performance.

Casey uploaded a video on about it on his channel and in that video he explained the rules of clean code and started breaking them and showing their effect on the performance of a program. These rules are:

- Polymorphism ✅ Conditional statements / switches ❌
- Never know the internals of what you are working on
- Functions need to be small
- Functions must do one thing
- Never repeat yourself

By breaking those rules, he showed how the program consumed less CPU cycles and got faster.

2. Application architecture and latency

Giovanni and Casey then discussed about the limits of performance in modern day applications.

Giovanni's view on this is that nowadays, applications are IO bound, which means that applications speed depends highly on the trade-offs between their IO subsystems, such as the disk or peripherals. He argues that, for example, if we have a very large database that is very slow to access, improving the code's performance will not make such a difference.

Meanwhile, Casey states that he does not believe that applications are IO bound, but rather people build them to be IO bound. Such as webpages that send small pieces of information to the server and then it returns another small piece of information repeatedly. He also reasoned that just because one part of the application depends on another is slow, that does not mean we should not work on efficient code because then we are condemning the application's performance never to be improved.

3. Clean code and information hiding

Information hiding is one of the key aspects of clean code, but this can cause many issues related to performance. For example, when hiding information with big class hierarchies the compiler must call a lot of accessor functions, this is not the case with exposed data types. If the compiler can easily access the definition of the class the performance is improved. Also, big class hierarchies are difficult to vectorize, making them difficult to optimize.

Issues with this kind of programming: You may have to work with code in a library that is not yours. You can always accept that you cannot touch the code of the library and live with its

performance, or you can open the library and extract implementation details or expose data types from the library to optimize the performance.

Information hiding may make the code difficult to understand, since when everything is working correctly then you do not have to see the code but calling a method that does not work as expected then looking for the piece of code that you want to check to understand the error harder.

4. Object Oriented Programming

Casey Muratori has one main issue with object-oriented programming and clean code: class hierarchies. He believes that they not only affect the performance of the code but also bloats the number of lines of code for the amount of operations that are being performed.

He also believes that object-oriented programming does not actually provide some of the benefits supporters of this type of programming claim it does, most notably readability, as he says that ORP is not more readable than procedural programming since by extracting functions with repeated code, good names for those functions... you can achieve this level of readability and also be much more efficient and smaller in size.

For Muratori good code should read roughly as it performs.

5. Clean code against efficient code

In the middle of the interview, Giovanni asks Casey Muratori what he thinks about clean code under the pretext that the code performs sufficiently well as to make final users content. Surprisingly, Casey has a very strong opinion in favour of performance and cites studies by "Big Tech" companies such as Google or Microsoft that would prove that better performance in a service increases customer satisfaction and engagement ad eternum. This is, no matter how efficient the code already is, any increase in efficiency will still improve the market value of the service until the increase in performance leads to 0 latency responses.

He mentions that any bussiness that prioritizes clean code will miss a great bussiness opportunity and states that "performance equals money".

6. Monoliths and microservices

Casey Muratori advocates for monoliths against microservices, stating that the latter are *"software engineering's solution for Conway's Law"*. He states that monoliths are usually noticeably faster than microservices and its main benefit (modules independent of each other) is negated by the fact that many times the result product is a *"mesh of services that strongly depend on each other"*. That said, he also argues that microservices is not a bad architecture in the case that the work division is unenforceable in any other way and if the code quality and speed trade-offs are considered.

7. Decisions against trade-offs.

Casey Muratori argues that decisions are not the same as trade-offs:

- Trade-offs are informed decisions in which their own impact has been already considered and accepted beforehand.
- Decisions are all other decisions which have been accepted and their impact has not been considered beforehand.

He places the example of using Python as opposed to a JIT-compiled language[1] or Cython. The former is the official and most used implementation of Python and is interpreted, while the latter languages get a noticeable performance difference when compared to Python. He states that if someone wants to start a new Python project, they should be aware of the language's limitations and ask themselves if the language is the correct choice, the amount of code written in that language, whether they are sure of that amount, etcetera. He also uses the example of Facebook and PHP, remarking that the former had to create a special compiler just for their own version of PHP and to get optimized code.

8. Premature optimization.

When asked about common pitfalls regarding good code, Muratori answers that premature optimization is something that is usually avoided with Donald Knuth's famous quote (and misused in this sense) about premature optimization being the root of all evil. He argues this has often been interpreted as simply

---

Don't worry about writing non-efficient code.

, which is not the actual meaning of the whole quote, as it has been taken out of context. The actual meaning in the context of the paper it was published would be something along the lines of the following statement:

If you have some source code of, say, 1000 lines, that is not as fast as it should be and 30 of those 1000 lines are executed more often than the rest, then you should focus your attention on those 30 lines.

He further argues that coding should be done considering which parts will be performance-intensive, and that performance should be considered since day zero (including architecture design) to obtain an optimal one.

---

[1] Python does have a JIT-compiled version in PyPy, and a JIT-compiled opt-in option of Python is slated to arrive with Python 3.13.

9. Other questions

Casey also states in the interview that he does not think we have any objective measure of code quality beyond performance measurements. He believes that things like clean code are based solely on subjective, speculative measures and that consciously engaging in practices that go against objective performance is “truly a trade-off” while doing so out of ignorance is just “making an excuse”.

He also disagrees with the notion that clean code improves cognition and reduces development time, mainly for two reasons. The first one is that it is something that is not comparable, as there are almost infinite ways to write a code that does the same thing and maybe there was a way of writing high performance code with the same level of cognition. Secondly, in the real world, a lot of times code must be rewritten many times in a short span of time precisely because of performance reasons, which means that prioritizing cleanliness over performance will just make developers rewrite code more often.

Finally, when asked what a good way is to not forget about performance considerations, Casey states that programmers should include performance education as a requirement for them as developers and affirms that it is easier than understanding how modern languages work, as CPUs (although modern ones are getting more complex) have a design based on simpler principles than languages.