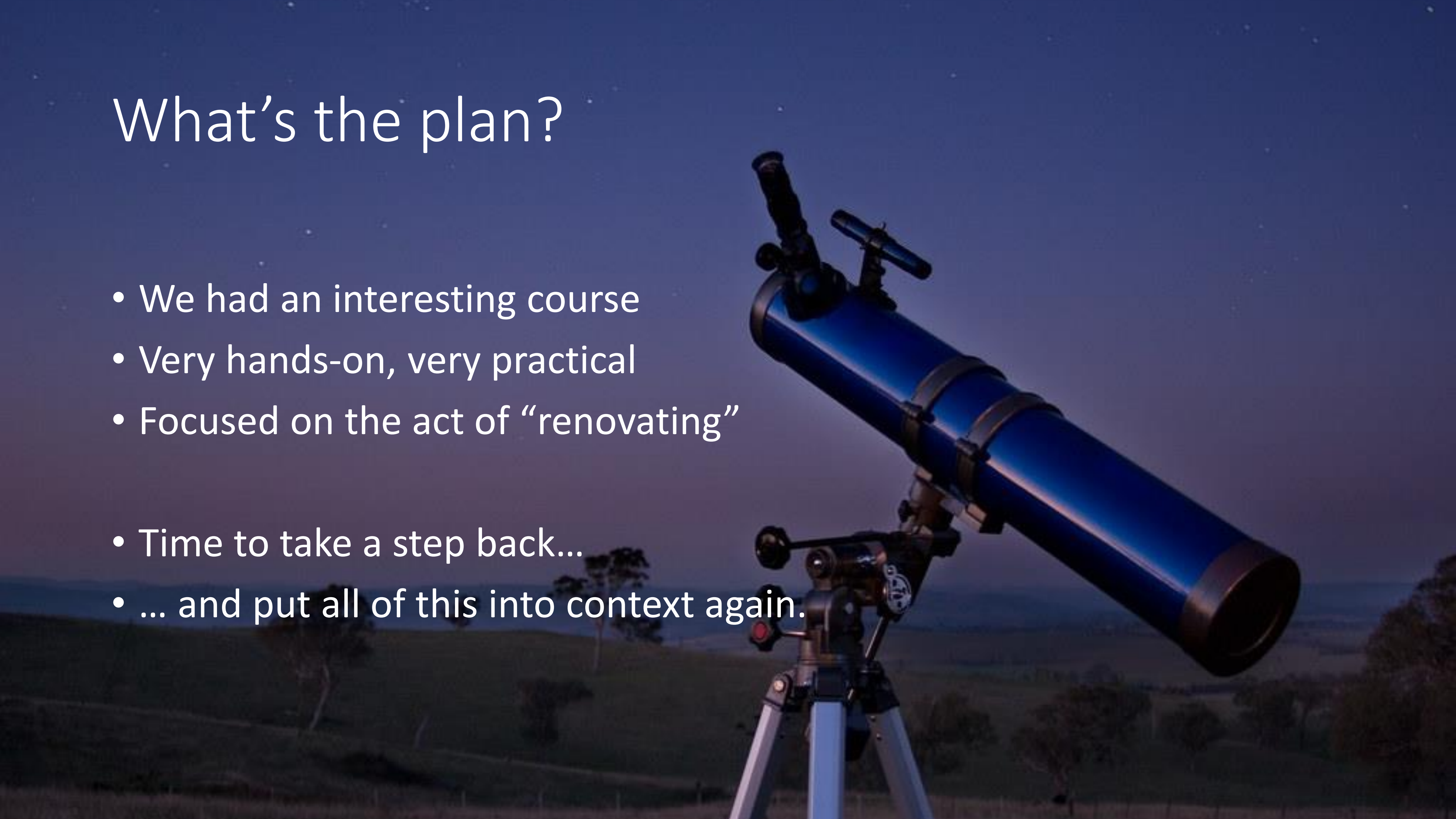
Legacy Code

My Perspective

Markus Doggweiler, CSS Insurance, October 2021

What’s the plan?



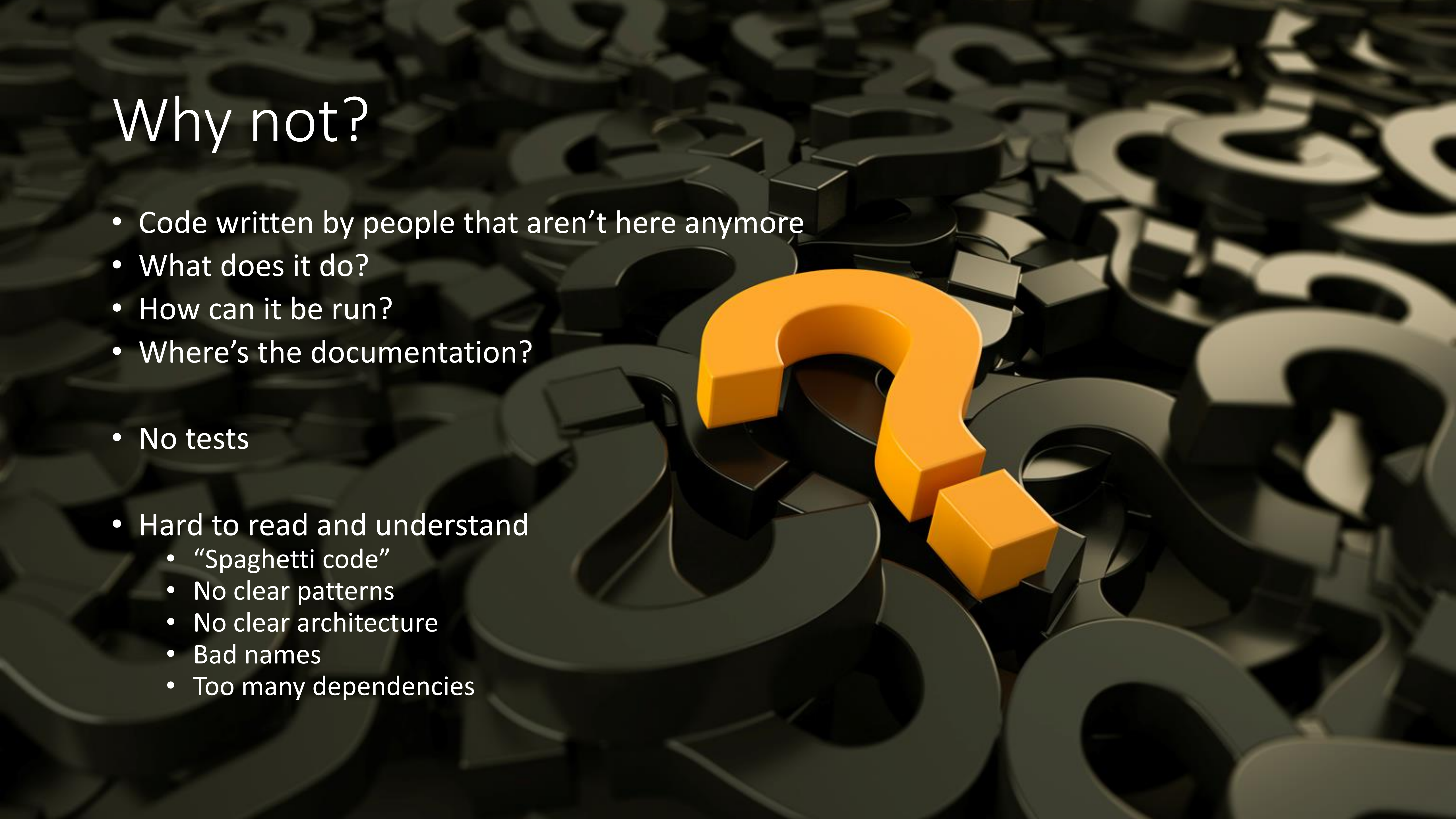
- We had an interesting course
- Very hands-on, very practical
- Focused on the act of “renovating”

- Time to take a step back…
- … and put all of this into context again.

Legacy Code – What are we talking about?

- In computing, a legacy system is an **old** method, technology, computer system, or application program, “…” **yet still in use**. (Wikipedia)
- Code **without tests**. (Michael Feathers)
- Code **is legacy** code **as soon** as it's **written**. (Unknown source)
- And there’s probably a lot more…

Legacy code is code that nobody wants to touch.



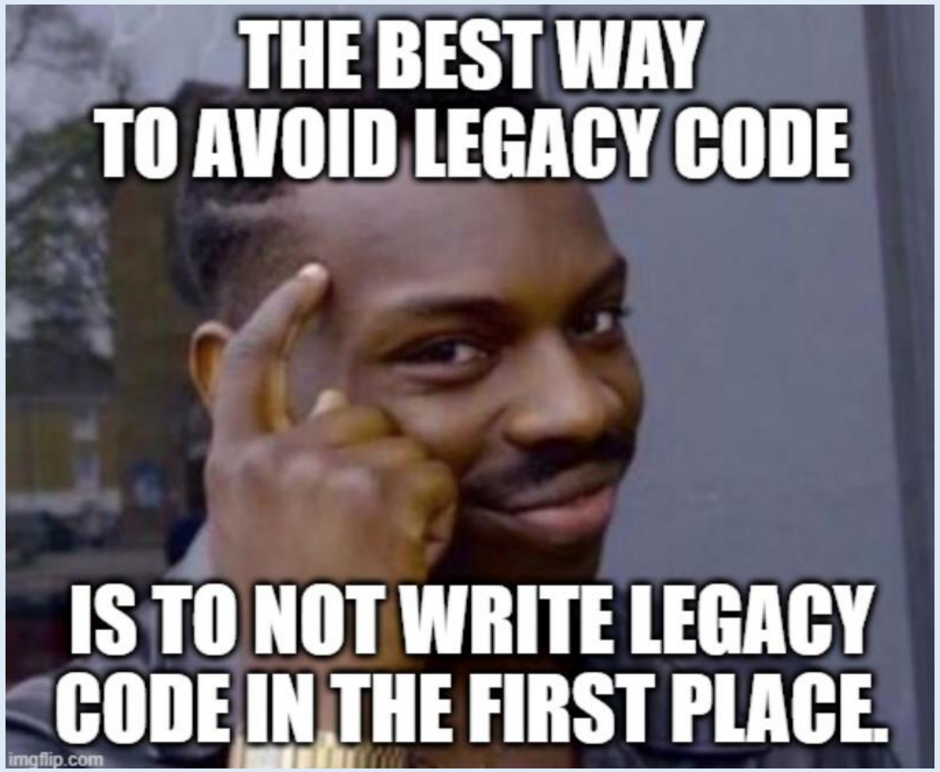
Why not?

- Code written by people that aren't here anymore
- What does it do?
- How can it be run?
- Where's the documentation?

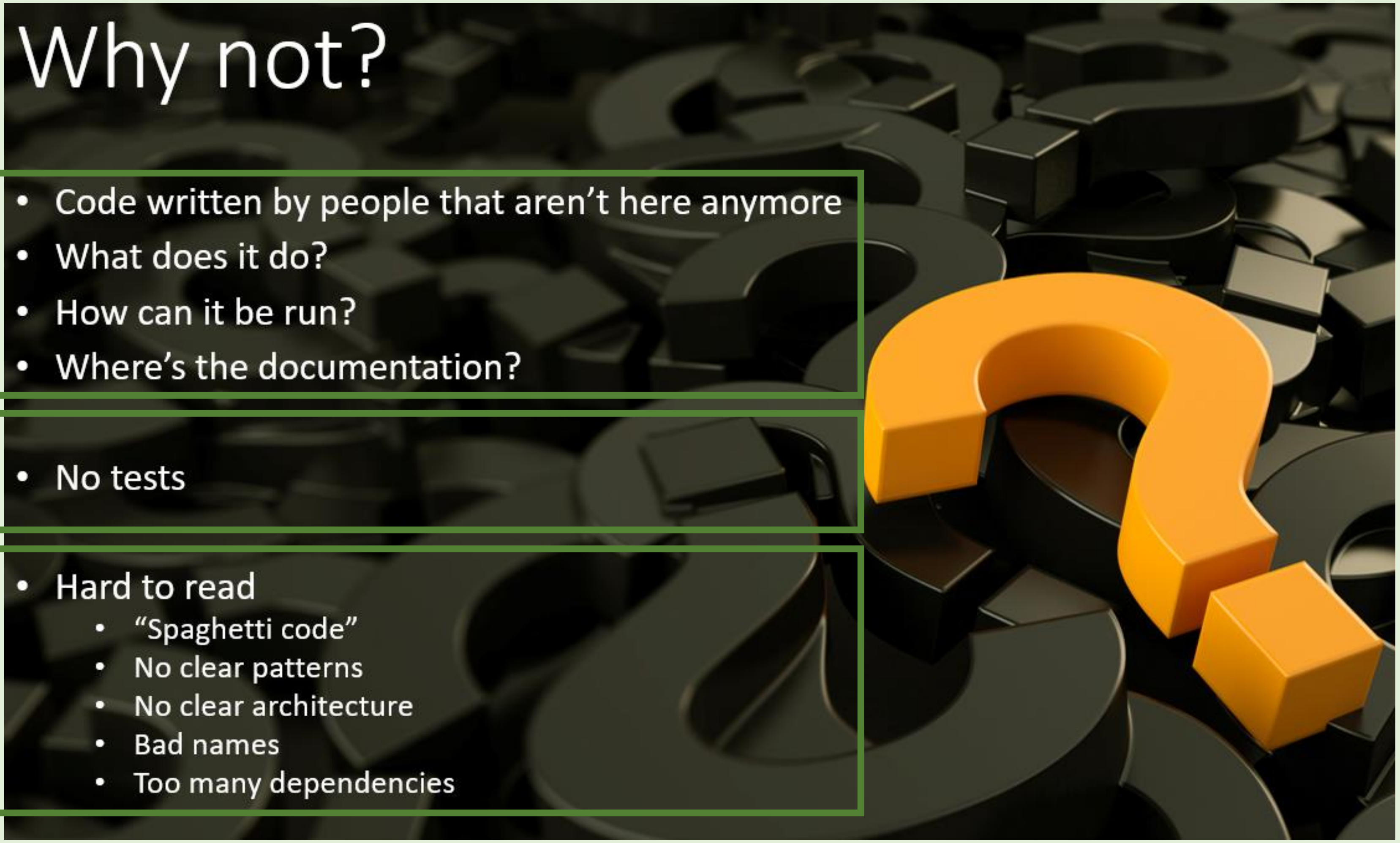
- No tests

- Hard to read and understand
  - "Spaghetti code"
  - No clear patterns
  - No clear architecture
  - Bad names
  - Too many dependencies

Ok, so what do we need to do?



But how?



Why not?

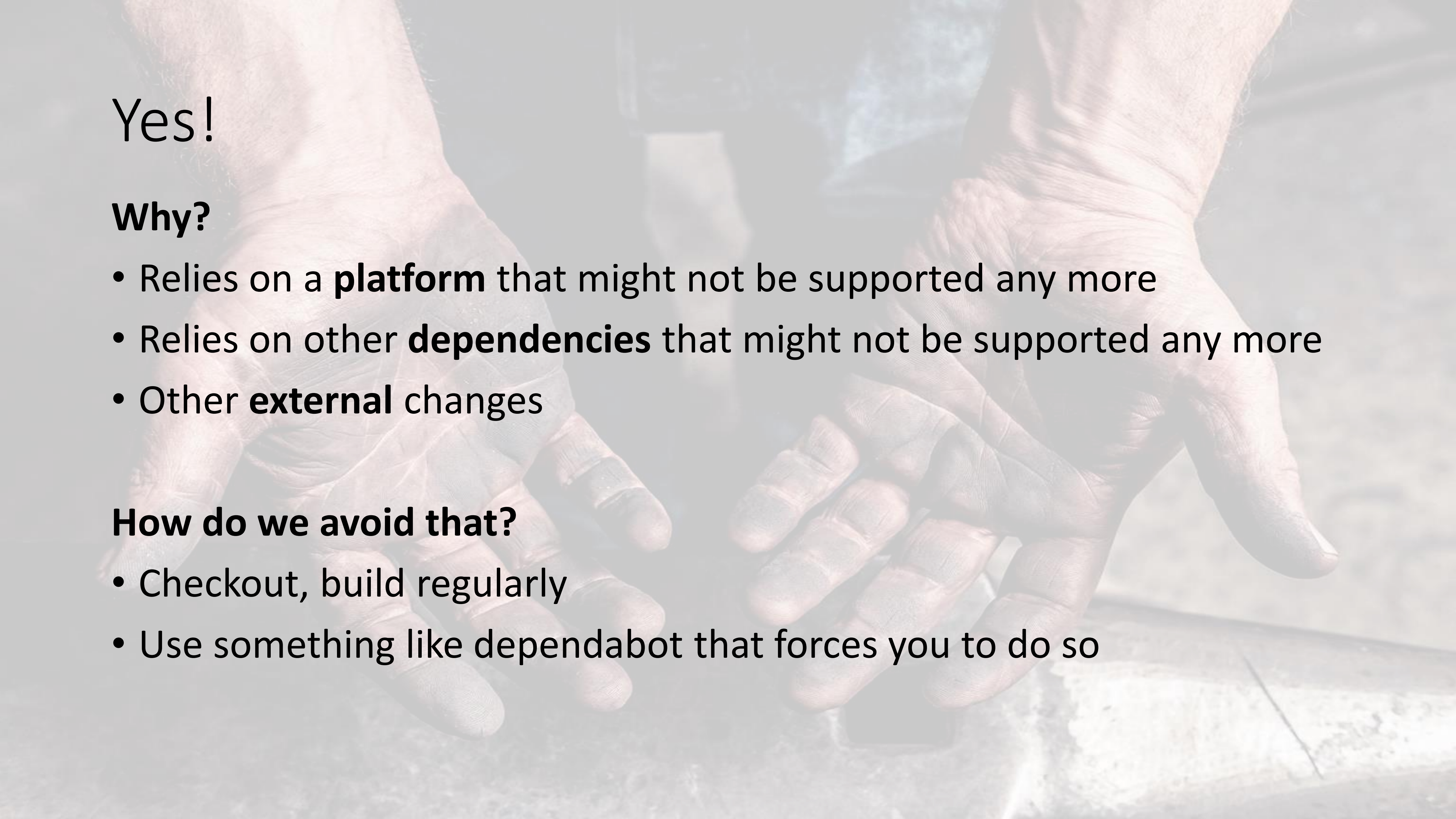
- Code written by people that aren't here anymore
- What does it do?
- How can it be run?
- Where's the documentation?

- No tests

- Hard to read
  - "Spaghetti code"
  - No clear patterns
  - No clear architecture
  - Bad names
  - Too many dependencies

Documentation	Code written by people that aren't here anymore; What does it do?; How can it be run?; Where's the documentation?
Automated tests	No tests
Clean Code	Hard to read

Can “good and clean” code become **legacy** just **by aging**?



Yes!

Why?

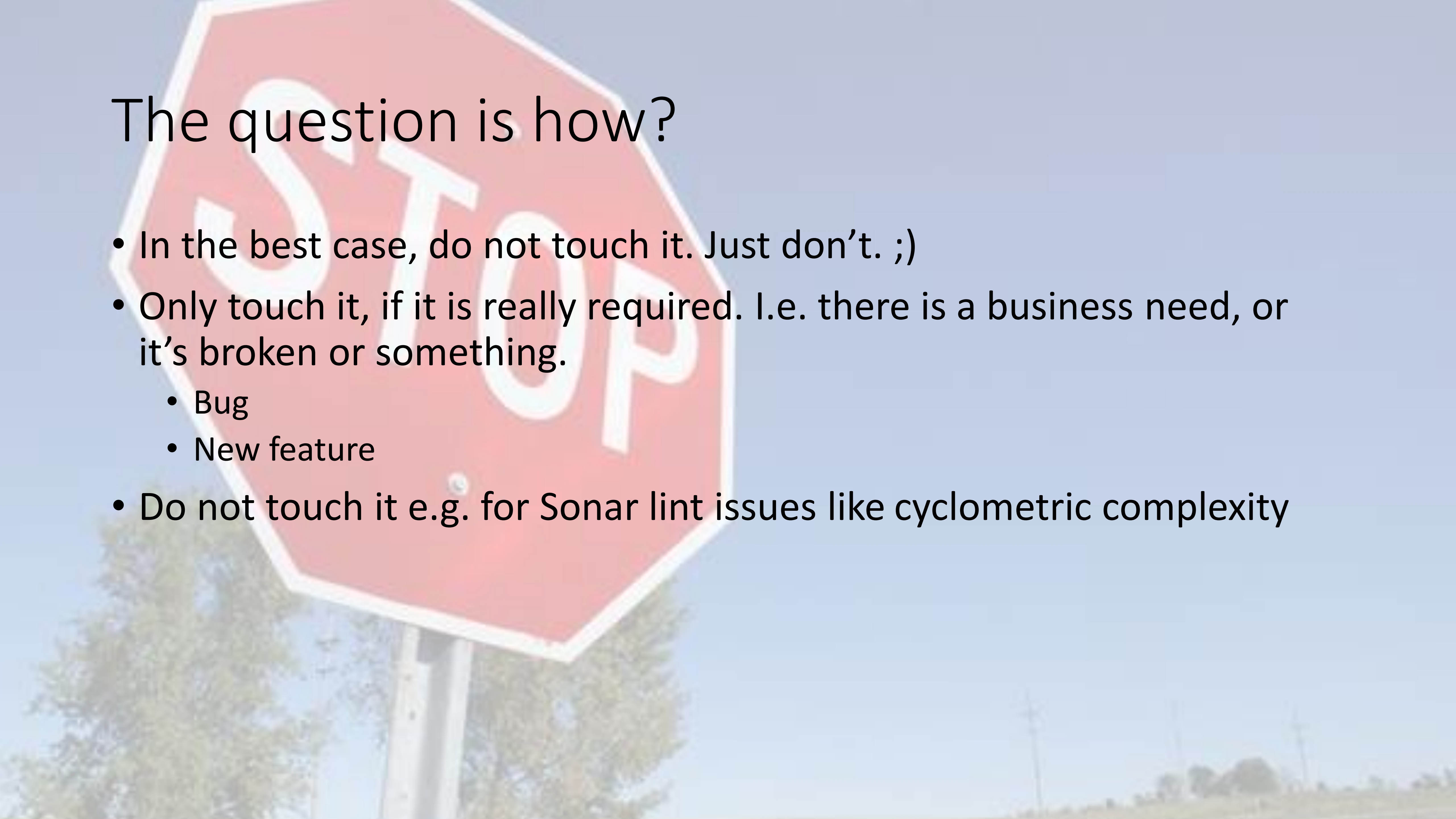
- Relies on a **platform** that might not be supported any more
- Relies on other **dependencies** that might not be supported any more
- Other **external** changes

How do we avoid that?

- Checkout, build regularly
- Use something like dependabot that forces you to do so

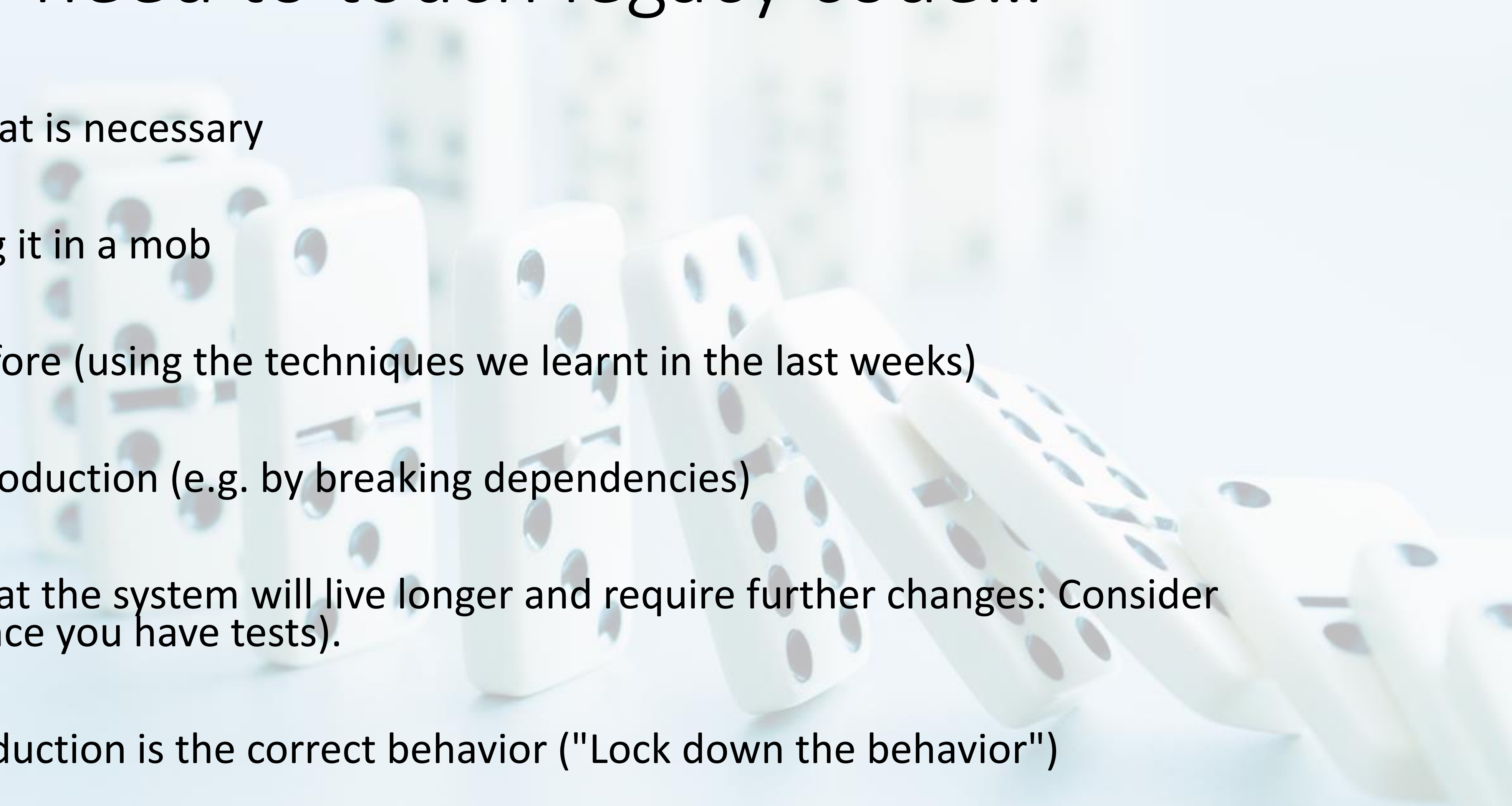
Anyway, there will always be legacy code…

So we need to be able to **deal with it**.



The question is how?

- In the best case, do not touch it. Just don’t. ;)
- Only touch it, if it is really required. I.e. there is a business need, or it’s broken or something.
  - Bug
  - New feature
- Do not touch it e.g. for Sonar lint issues like cyclometric complexity



If you do need to touch legacy code…

- Only touch what is necessary
- Consider doing it in a mob
- Write tests before (using the techniques we learnt in the last weeks)
- Don’t break production (e.g. by breaking dependencies)
- If you know that the system will live longer and require further changes: Consider refactoring (once you have tests).
- What is in production is the correct behavior ("Lock down the behavior")

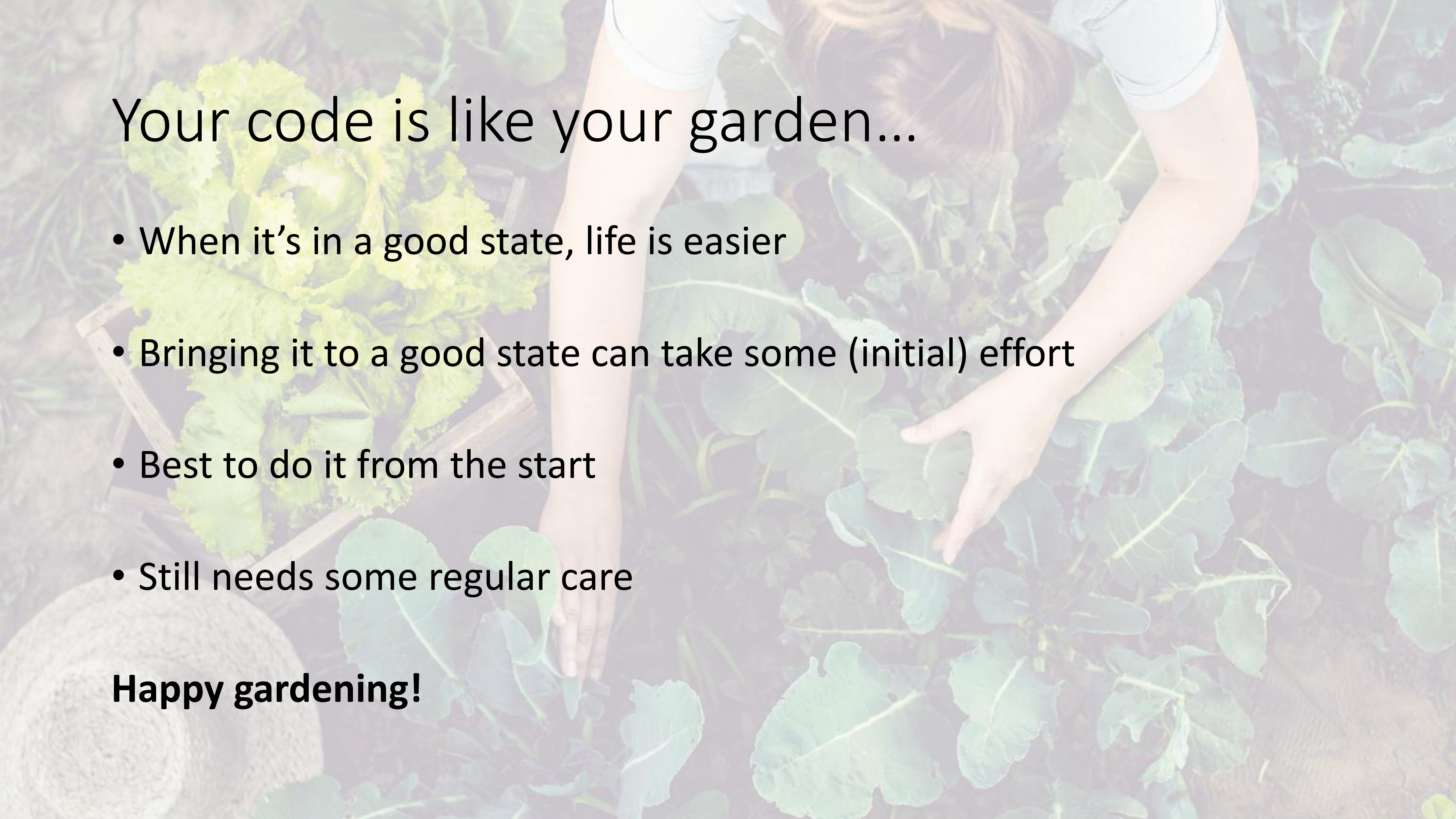
Personal example from my last employer

- We had a so called “expression engine”
  `DateTime.Now.AddDays(10).ToString()`
- The parsing functionality had bug
- We knew it
- But we didn’t change it

Most important thing when touching legacy code…

… is to apply what we learnt:

- Write tests
- Don’t break production
- Break dependencies
- Change only what is necessary



Your code is like your garden…

- When it’s in a good state, life is easier
- Bringing it to a good state can take some (initial) effort
- Best to do it from the start
- Still needs some regular care

Happy gardening!

Thanks for your attention.

- Questions?

- Discussion?

Or contact me later at markus.doggweiler@gmail.com