Being a developer

the non-technical part

Who am I

Edwin Hermans

CTO at LabBox, a startup studio in Brussels incubating projects around mobility

- From developer into Analyst, Devops, IT director, general manager, and finally CTO
- Formerly founded & directed a web development agency in Brussels
- Consultant & Trainer for a few corporates (Sodexo, Adecco, McKinsey, …)

The following slides are personal opinions

Part I: Soft skills

Soft skills

The obvious ones

- Passionate
- Curious
- Social / team player

Adaptability

- Languages are living things
- Patterns come & go
  (SQL, MVC, procedural, …)

Ability to contextualize

- Quality of code VS deadlines
- Priorities
- Project lifetime

Ego-free

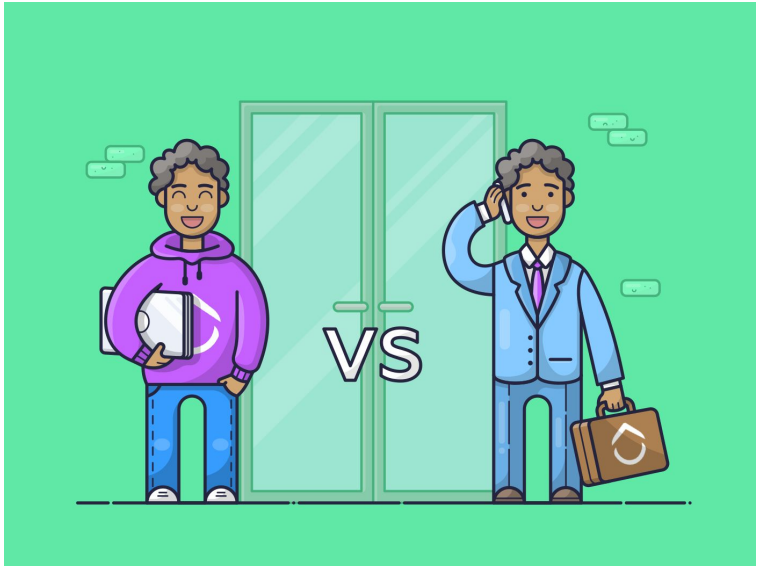
- Share, both ways
- Challenge
- Be challenged

Part II: Mindsets

Different worlds

Startup

- Discovery & innovation
- Short-term vision
- Reduced processes



Corporate

- Long-term mastered projects
- Structured processes & phases

Startup world

The objective is to
validate assumptions

- Build first, think later
- Build what's needed and only what's needed
- Trash anything that’s not relevant anymore

Corporate world

The objective is to
build a stable, performant, and maintainable product

Analysis phase

- Functional analysis
- Technical analysis

Implementation phase

- Coding
- Automated testing

Quality control

Part III: Methodology

across all mindsets

Top-notch development environment

- **Get an easy setup & running local environment**
  Don’t be afraid when you need to work again on an previous project

- **Get a debugger**
  Get yourself comfortable to understand what your code is doing

- Make it easy to replicate everything - especially bugs - without screwing production

- Make it easy to deploy often

Organisation

- **Tickets!**
  And super easy to create and maintain

- **Even alone!**
  All the time you'll find yourself thinking of stuff to do while in the middle of something else

Versioning

- **Regular git commits - also even alone**

    a. It (should) explain the why
    b. Allows you to ask yourself if your new code is understandable

Documentation

- **The why, not the what**

- **Document what can't be understood by reading the lines one be one**
  a. I can see what this function does, but why do we need that
  b. We know there's a more classic way to do it, why this?
  c. Big picture of unique flows or concepts
  d. Big picture of stack

- Document how to setup your project from scratch, which scripts does what ...

Testing

- **Unit testing**
  When there is a complicated business logic algorithm

- **Functional testing**
  To ensure your new code doesn’t break anything

- **Test driven development VS testing what needs it**

Thanks :)