PAUL M FOX

Programmer, software ontologist and hero.

Brighton, UK | p@yhp.io | +44 7429 142570

I've been building software applications for 20 years.
Now I'd like to help you change the world.

Listen up, because we don't have much time. In life, I mean. But it's not all bad. The assets we share on this journey of ours are some terrific tools (like computers and behavioural enconomics), reciprocal altruism and a boundless capacity for innovation.

You could say my job is to be a kind of digital Sherpa; a sort of technical life coach who'll take you from the 'what' all the way up to the peaks of 'why' and back down into the valley of 'how'. Saying that would be technically inaccurate, though, because I don't have a *job* exactly.

For me, work is play and life is thinking about process. There are no ends, there are only means. Take my hand: things are about to get weird and brilliant. But do it quickly, because we don't have much time.

Quick references

https://github.com/yourheropaul/

https://www.linkedin.com/in/yourheropaul/

http://yhp.io https://endian.io

A condensed timeline of recent work

now

Joystream (Oslo, Norway)

Product manager and lead engineer. 6 months so far.

Joystream is a video platform controlled, owned and operated by its users. It's YouTube backed by a blockchain, powered by **Substrate** (implemented in Rust) with an awesome **React** (powered by **TypeScript**) frontend and an **IPFS** storage layer.

2019

Kowala (Nashville, Brooklyn, Grand Cayman)

1 year Lead developer, 2 years hands-on CTO.

An ambitious attempt to create a family of autonomous stablecoins. The client was a fork of the **Go-Ethereum** project with a novel consensus mechanism, discovery protocol and EVM. The wallet was built in **React Native**, the business logic was comprised of a set of **Solidity** contracts and there was a cast of supporting **Go** tools.

The project was distinguished from many of its peers by its completely test-driven engineering approach. It has complete, **Dockerised** **end-to-end tests** for every part of the system, including the blockchain and network. These tests were run on every code commit. The code is available on GitHub.

2016

Endian (UK, Portugal)

Founder. 4 years of good times and counting.

A collective supported by an agency and consultancy. We help startups define their goals, find their teams, build out their products, and expand to the stars (not literally, yet).

Our founders are technical, and we like to embed ourselves in our clients' businesses as developers or CTOs. We lead from the front, and we build our own products whenever we can.

We build everything in **Go**, **Elixir** and **React** via **TypeScript**. Our daily methodology is a combination of **Agile** and **BDD** for engineering, and a combination of skepticism and stoicism for business.

2015

Niwa (Spain, Stockholm, US)

1 year Lead developer.

An **IoT** application focused on growing plants. The product is a an automated, home garden. The firmware for the chip is written in **C++** (specifically **Wiring**, via **Particle**) and **ARM64 assembly**, the backend APIs are **Go** and **Node.js** backed with **MongoDB** and later **Postgres**. The end-user mobile app is written for **Ionic 2**, which uses **Angular 2** via **TypeScript**.

The entire system, *including the chip itself,* has fullt automated end-to-end tests. For the hardware this is accomplished by a custom-built testing rig.

2014

Homemade (London, UK)

1 year software engineer, 2 years lead developer, 2 years platform architect.

Homemade Digital is a London fin-tech agency with big clients in the third sector. They build very large donations platforms in **Go**, **Node.js**, **PHP** and anything else that's necessary. Most of their applications get very large amounts of traffic over a short time (they're essentially telethons) and so scaling technologies like **GCP** and **AWS** are typically used to deal with the load.

2010

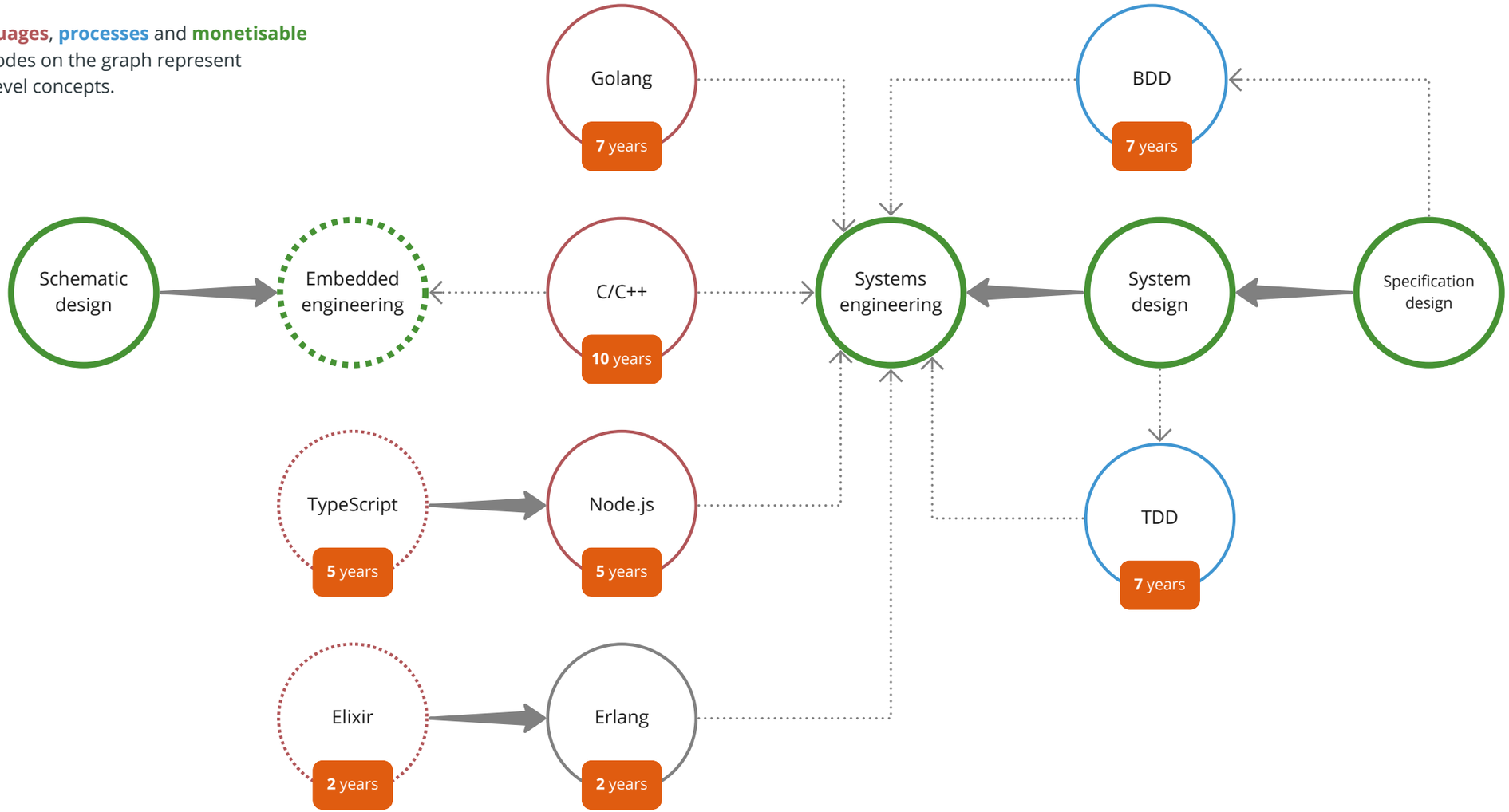
A1

PAUL M FOX

High level map of programming skills

This is a relational map of things I can do that will likely be useful to you in backend capacity. We're focused here on the lowest level of utility -- actually writing code -- though as we'll see, a quality engineer needs a much more comprehensive toolbox.

Featured here are **languages**, **processes** and **monetisable applications**. Dotted nodes on the graph represent abstractions on lower-level concepts.



A2

PAUL M FOX

My ideal code-testing and quality assurance flow

High quality code is written by one or two developers, and then vetted through five automated processes, one peer review, and an end-user verification test.

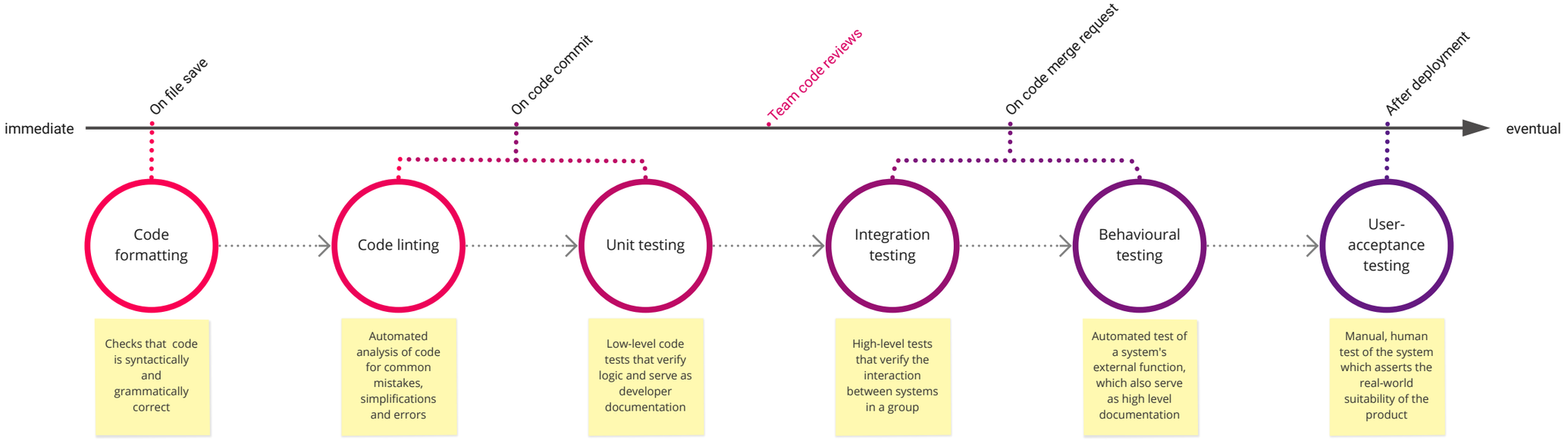
The processes are run in sequence. Each process is more computationally (and in the case of UAT, often financially) expensive and takes longer, so when one fails, the vetting process is aborted and the developers should be alerted.

The goal is to make error feedback loops as short as possible. The longer developers have to wait to learn about potential problems, the slower their work.

A quick note on code quality: the only real measure of code is its utility. That is, code isn't intrinsically *good* or *bad;* it's *useful* or *not useful*.

The difficultly with measuring code usually comes from the definition of *utility* itself. Certainly, the code has an intended ultimate purpose, but what else will be done with it?

I'm assuming here that the code will be read, maintained and possible extended in the future, and I consider those to be secondary uses. Hence the linting, formatting and low-level tests outlined below.



A3

PAUL M FOX

Why I'm so interested in feedback loops

The exact amount of time each process consumed is highly dependent on the tools used and the team configuration, but it's generally in the order of milliseconds through to hours.

Short delays in error feedback increase the chance the developer can fix issues without losing focus; long delays decrease the chance.

The optimal feedback arrangement is illustrated in the flowchart below. Modern tooling, especially code format assertion, continuous integration servers and virtualisation, make this process easy and efficient to use and set up.

This all makes a significant difference. The shorter the average delay, the faster the build.