

Ruff: 一次破局 IoT 研发困境的尝试

郑晔



1

IoT 进展缓慢

硬件研发困局

产品经理与硬件工程师难于协同

瀑布式研发

重复造轮子

系统与应用一体

软件人的“硬件”鸿沟

硬件词汇表

- GPIO
- I2C
- C
- 驱动
- 时序
-

软件词汇表

- 需求
- 用户体验
- 设计模式
- 软件系统架构
- 高可用性
-

软件和硬件

	软件工程师	硬件工程师
关注点	产品、用户体验	功耗、实时性
理念	试错	预先定义
交付方式	迭代交付	一次性交付

2

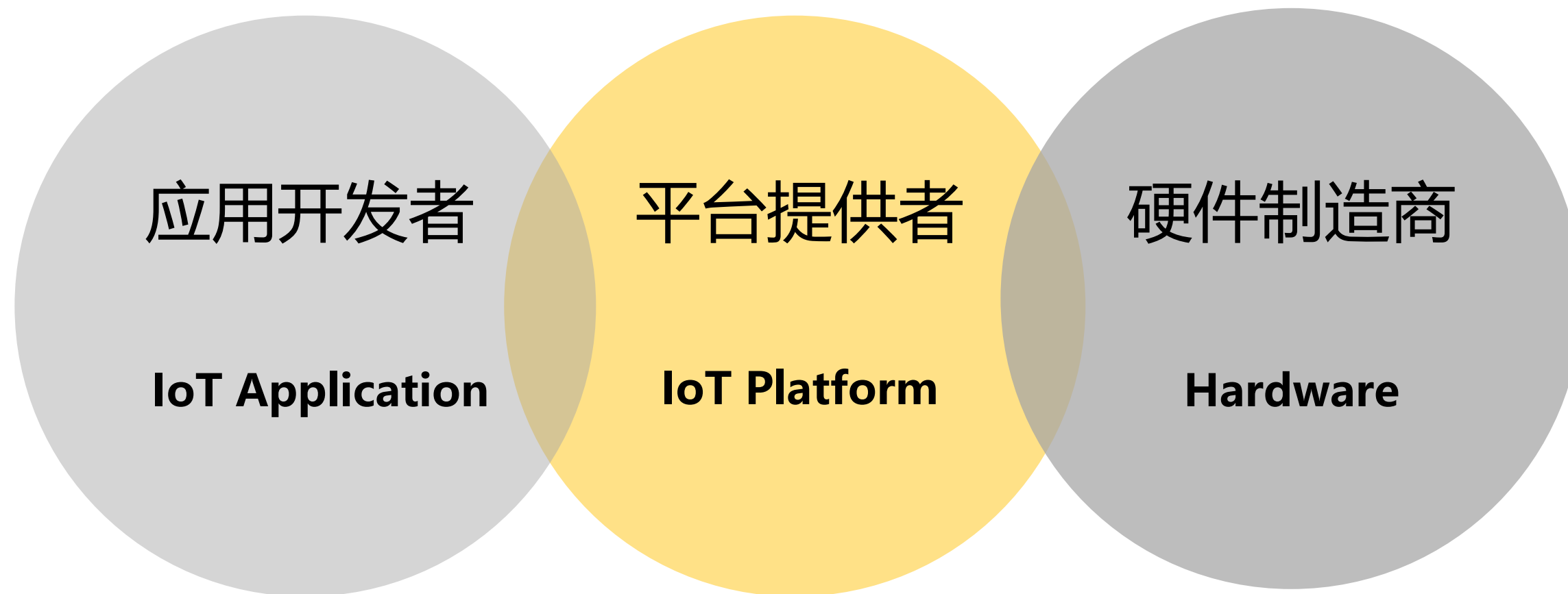
IoT 应用研发问题何在？

硬件制造商在编写应用

“劳动生产力上最大的改进，以及劳动时所表现的更多的娴熟程度、技巧和判断力，似乎都是分工的结果。”

——亚当·斯密 《国富论》

IoT 时代的社会分工



3

Ruff 一次 IoT 平台的尝试

IoT 平台衡量标准

现代程序设计语言

面向应用的抽象

提供生产支持

程序设计语言

传统的 C/C++

编程风格

C 语言原生只对过程式编程有支持

自动内存管理

缺少自动化内存管理能力，普通程序员经常会犯一些低级错误，造成程序崩溃

可移植性

缺少可移植标准，交叉编译是行业痛点，面对不同的硬件，需要花费大量时间让代码在运行起来

包管理

缺少包管理能力，不同的程序员会反复构建类似的代码，造成行业的浪费

测试

对于测试缺少内建的支持，测试的编译运行时间会随着代码规模而不断增长，没有小步开发的基础

JavaScript

编程风格

支持面向对象和函数式编程等多种现代编程范式，开发者可以根据需要自行选择

自动内存管理

支持 GC，开发者无需顾忌内存

可移植性

程序可移植，底层差异由运行时屏蔽

包管理

NPM 软件仓库，几十万个软件模块，开发者按需取用，无需重复造轮子

测试

有多种测试框架，开发者可以很容易在开发环节中进行测试

其它语言

运行时

如 Python、Java 之类的语言，很少有能对 MCU 进行支持的运行时，这使得它们顶多能做到原型级别开发，而无法深入

流行度

类似于 Lua 这种本身运行时很小的语言在嵌入式环境中也有应用，也确实有一些项目做到了，比如 NodeMCU，但 Lua 目前尚属小众语言，其前景取决于行业发展状况

面向应用的抽象

硬件抽象是需要的

汇编语言——C 语言——Java 语言

摩尔定律让硬件性能越来越强大

传统方式

```
GPIO.output(11, GPIO.HIGH)
```

应用抽象

```
led.turnOn();
```

使用应用开发者的语言

抽象级别

	特点	代码示例	典型平台
无抽象	面向硬件接口编程	<code>GPIO.output(11, GPIO.HIGH)</code>	大多数 IoT 平台
编程接口	开发者使用软件抽象, 屏蔽底层硬件接口	<pre>board.on("ready", function() { var led = new five.Led(13); led.strobe(); });</pre>	Tessel、Jonny-Five、Cylon.js
硬件配置	将硬件配置进行隔离, 让开发者不必关注配置细节	<pre>\$.ready(function (error) { \$('#led-r').turnOn(); });</pre>	Ruff

提供生产支持

生产支持

原型开发只是第一步

支持 MCU 是重要的能力

有了抽象的支持

分离应用与硬件配置

跨硬件的应用

分离应用与硬件配置

应用开发者无需关注硬件如何配置，
可以将更多的将注意力放在应用逻辑本身

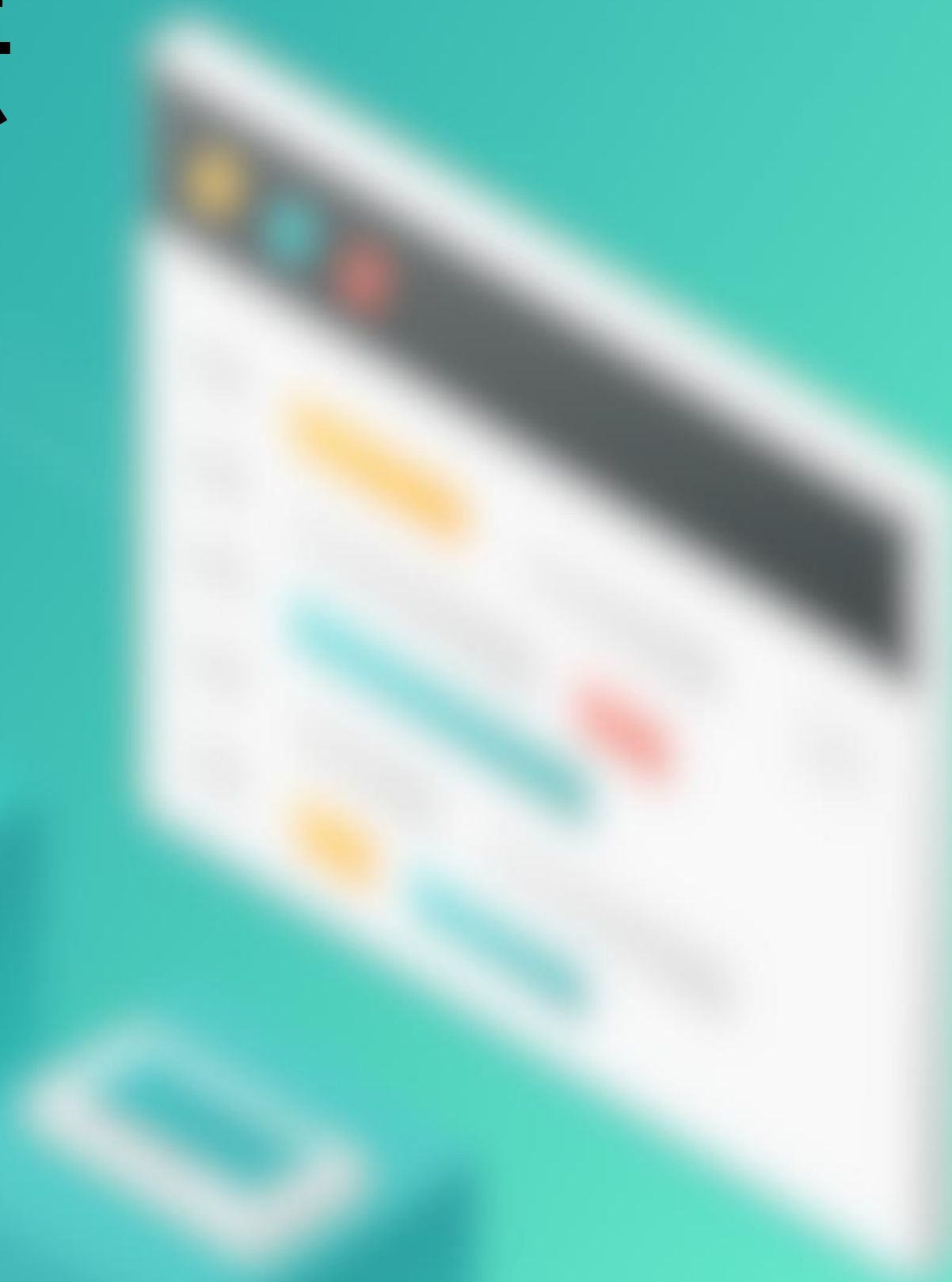
硬件具体的配置方式可以在具体的部署时实施

Ruff 的做法

现代程序设计语言：JavaScript

面向应用的抽象：`led.turnOn();`

提供生产支持：支持 MCU，分离应用与硬件配置



开发模式的转变

可移植性

应用可以在不同硬件上移植，软硬件之间只需适配接口

研发模式

应用在开发时不必知道具体的硬件，只要在交付时，将应用部署在硬件上即可，实现了二者的分离，二者便可以各自独立发展

研发流程

研发和生产可以采用不同的硬件，在研发期采用既有硬件进行测试，在完成需求验证之后，再根据情况生产实际的硬件

迭代模式

硬件、系统与应用将成为三个独立的概念，可以用不同的节奏发布，迭代开发成为可能

4

问题?



ruff.io