

从经典软工到开源开发

周明辉

zhmh@pku.edu.cn

北京大学

关于软件开发

软件的复杂性：需求的复杂性



1990年4月10日，在伦敦地铁运营过程中，司机还没上车，地铁列车就驶离车站。当时司机按了启动键，然后离开了列车去关一扇卡着的门（正常情况下如果车门是开着的，系统就会阻止列车启动），但当门终于关上时，列车还没有等到司机上车就开动了。

2021年7月20日，郑州暴雨，地铁启动后发现积水暴涌，“列车长尝试把车开回海滩寺，但因为地铁的自动保护设计，开不了，铁轨上都爆出火花了。”

平原公子赵胜

7-21 11:04 来自 荣耀20 PRO

+

关注

#郑州地铁五号线#有媒体发了一篇郑州地铁五号线被困人员的口述报道，基本能还原整个过程：

- 1、大部分人20日下午出门时，并没有意识到会发生如此猛烈的降水和洪涝，很多人觉得地上积水也不多，就正常出门了。
- 2、口述者下午乘坐地铁回家，一路都有人上地铁，所有没有觉得有什么问题。
- 3、事故在海滩寺站和沙口站之间发生的，乘客能从车窗看到车外水急剧上涌，列车长尝试过把车开回海滩寺，但因为地铁的自动保护设计，开不了，铁轨上都爆出火花了。
- 4、水开始涌进车厢，当时的路段是倾斜的，车尾地势高，大家聚集在车尾，后来列车长尝试过组织自救，指挥大家往车头走，打开第一节车厢的门，通过列车内部的人行通道出去.....一部分人走了出去。
- 5、但是人行通道非常窄，人又多又挤，很多人就被迫退回了车厢，列车长只能关闭车门打电话求救，大家待援。
- 6、大家虽然有人恐慌、哭泣，但是还是保持了很好的组织和秩序，有个姑娘不断安慰大家，维持秩序，安抚情绪，劝大家不说丧气话，保存体力。还有人一直在通报救援队的情况，说他们已经到了外面，正在垒沙袋，拉绳子。
- 7、水位越来越高，车窗外已经一人多高，车厢里的水也渐渐漫过胸口、脖子，很多人体力不支、缺氧、低血糖，车厢里还有老人、孩子、孕妇.....很多人已经在发消息交代身后事了。
- 8、有人冲动要砸开车窗，被一位大叔制止，因为外面的水位更高，如果水涌进来，伤亡会更多。
- 9、出现了转机，因为水流的冲击，车厢发生了偏移，一边高一边低，往上翘的一面，车窗露出了水面，于是大家提议，砸开那块玻璃透气，在大家的努力下砸开了车窗，缺氧状况得到了明显的好转。
- 10、正在大家积极用灭火器继续破窗的时候，救援队出现在车厢外面。他们最先通过凿开的那处玻璃窗，将破窗器递到车厢内。车厢内里面大家也在接力将破窗器向后传递。救援人员把车长车厢打开了，大家排队陆续出去，孕妇、女人、孩子先走。
- 11、救援人员在外面拉，车厢里的群众互相帮助，一个个把同伴从水中托起来.....脱离险情后走了十来米，就到了安全的地方，大家互相搀扶着往前走.....但是还看到许多穿着制服的人向着反方向逆行。

总结一下，五号线被困地铁上的乘客、车长其实努力过自救，只是情况实在是太复杂太危急，才会选择原地待援.....这趟列车上的群众表现出来极高的组织力和整体素质，但还是不幸有12人死亡，5人受伤。

Brooks法则： 协同的复杂性

- 向进度落后的项目中增加人手，只会使进度更加落后。

- 《人月神话》

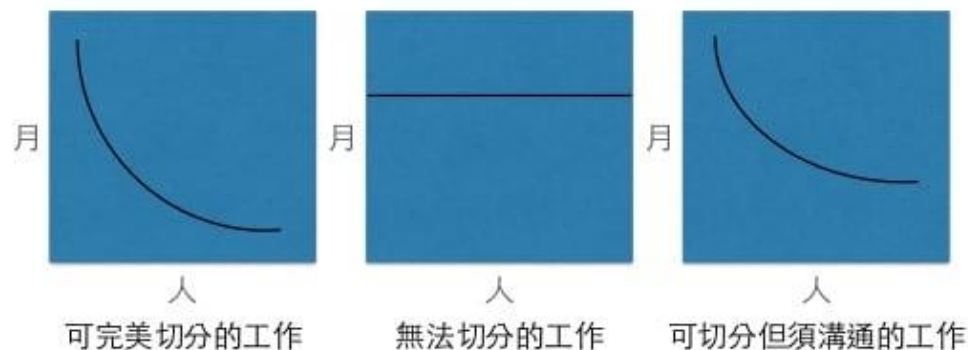
人、月 之外的變因

1. 是否可完全切分：

生小孩要9個月，10個媽媽生也是要9個月

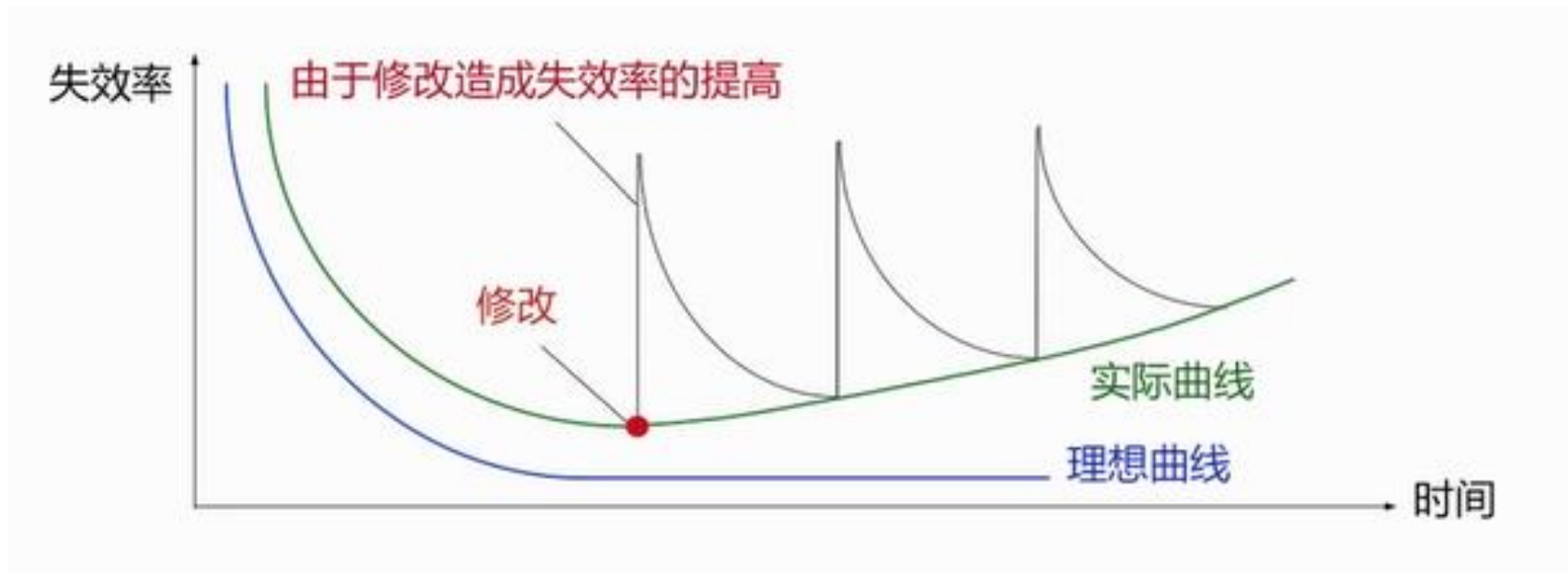
2. 是否需要溝通：

訓練成本與交流成本也應計入工作量



软件行业的演变：从制造产业到服务产业

- 软件工程伊始，软件开发被认为是制造业，但是，难以用销售价值决定软件价值，因为75%的资源都消耗在软件维护上。
- 因为软件维护的开销巨大，软件产业逐渐被认为是服务产业。



经典软件工程

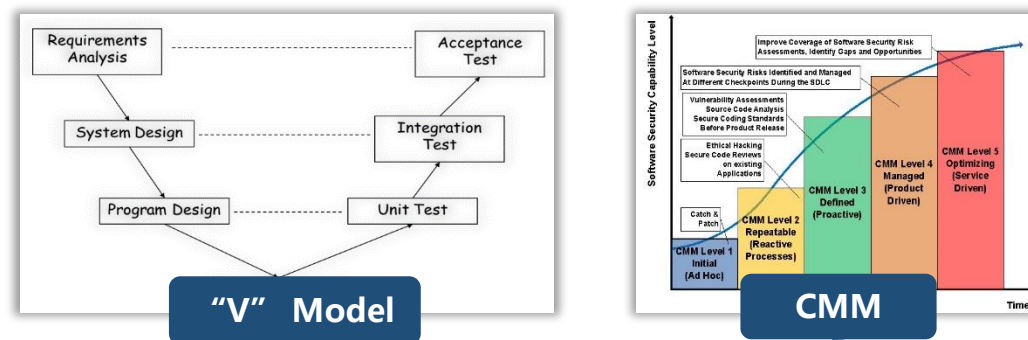
制造产业思想

“制造产业”的软件工程



NATO Software Engineering Conference 1968

向经典的工业生产学习
软件工程三要素：方法、工具、过程



研究过程管理方法：组织性

研究软件生产工具：自动化



测试自动化



编程与修复自动化



验证自动化

工程化开发方法：自上而下、逐步求精

元素		工程化开发方法 自上而下，逐步求精
理念	需求	有明确用户和明确一致用户需求描述，这是软件开发的前提
	质量	源代码满足需求描述的程度
	效率	开发满足需求软件的时间和成本
方法	过程模型	瀑布模型、敏捷模型、能力成熟度模型
	支持工具	软件产品线工具集
	计算环境	面向计算机环境

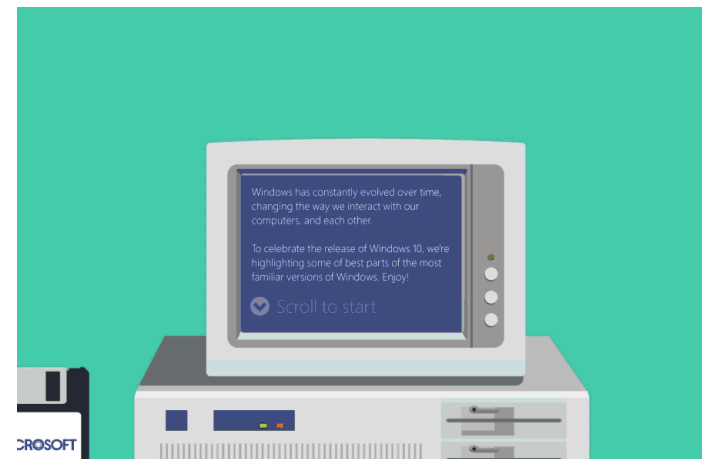
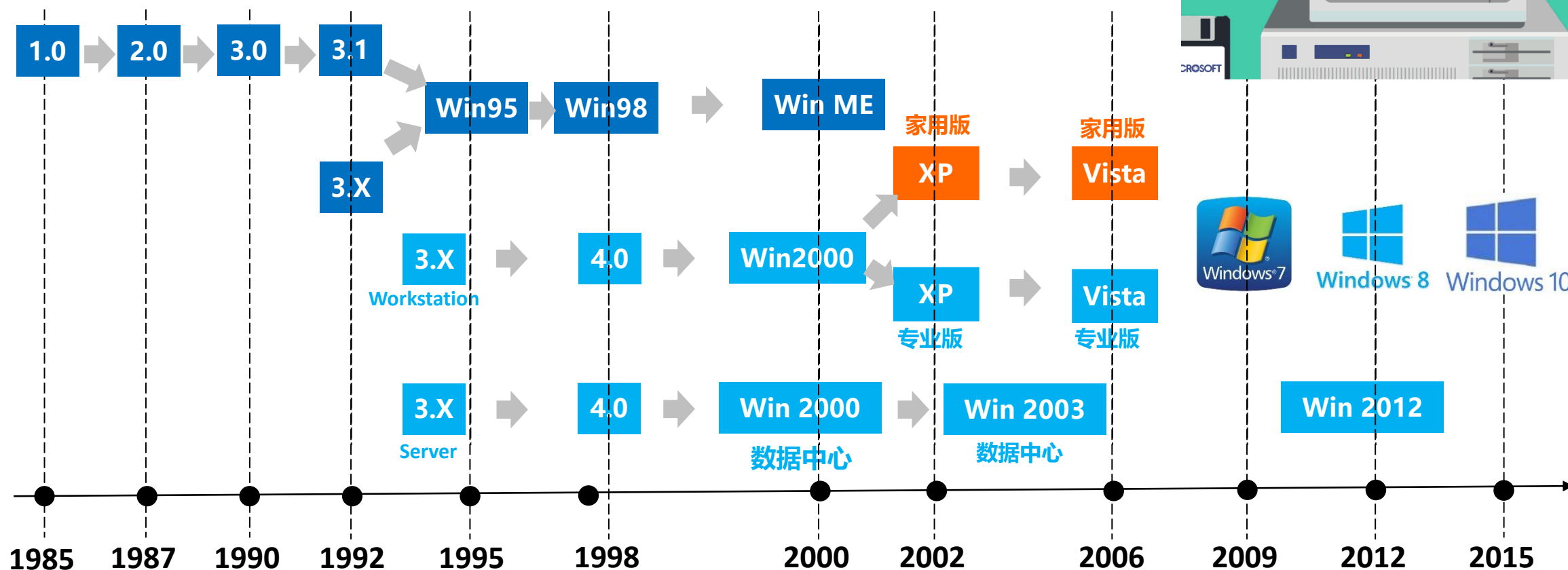
- 用户需求是确定的
- 可被理解的
- 可被表述的
- 可被线性分解还原的
- 严格定义的流程和时间

工程化的经典：Windows 演化史

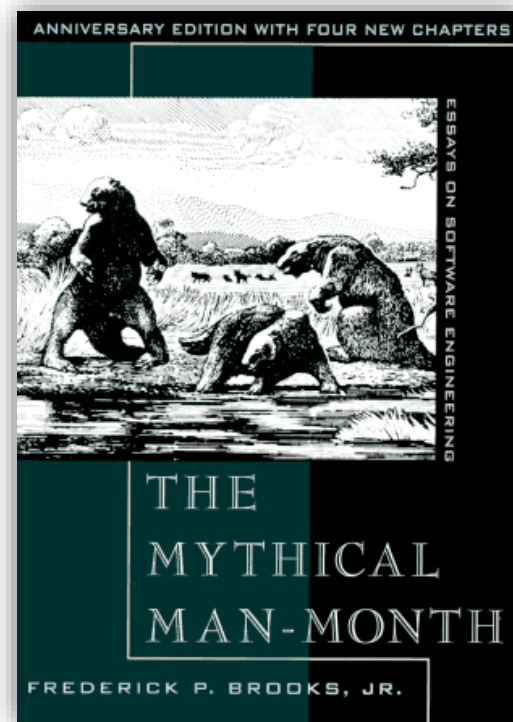
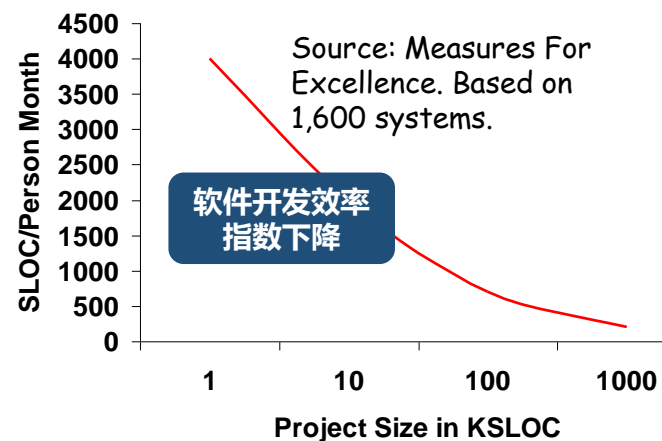
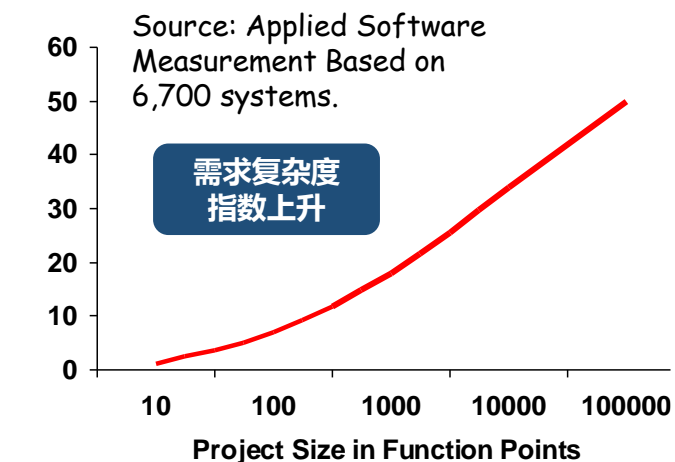
严格控制软件出现分叉版本

MS-DOS版本

NT版本



工程化开发方法的协同瓶颈



“No Silver Bullet”

软件开发管理如何突破群体协同生产的天花板？

工程化开发方法的领域瓶颈

复杂软件系统

人与系统的边界模糊了

分散控制

固有的冲突、不可知、
多样需求

失效是常态

持续演化和部署

异构的、不一致的和变
化的元素

经典方法的前提

人仅仅是系统的使用者，人的群体行为不为系统所关注，系统不涉及社会性的交互

求解过程是集中统一控制的、**自上而下的分解**过程

需求是能够被提前获知的，所有的冲突必须被解决，其变化是缓慢的，决策是稳定的

缺陷能够被剔除，失效应该极少产生

系统的改进是阶段性的

变化的结果是可以被充分预见的，部署配置信息是精确的，可被严格控制的，构件与人员是完全同质的

开源开发

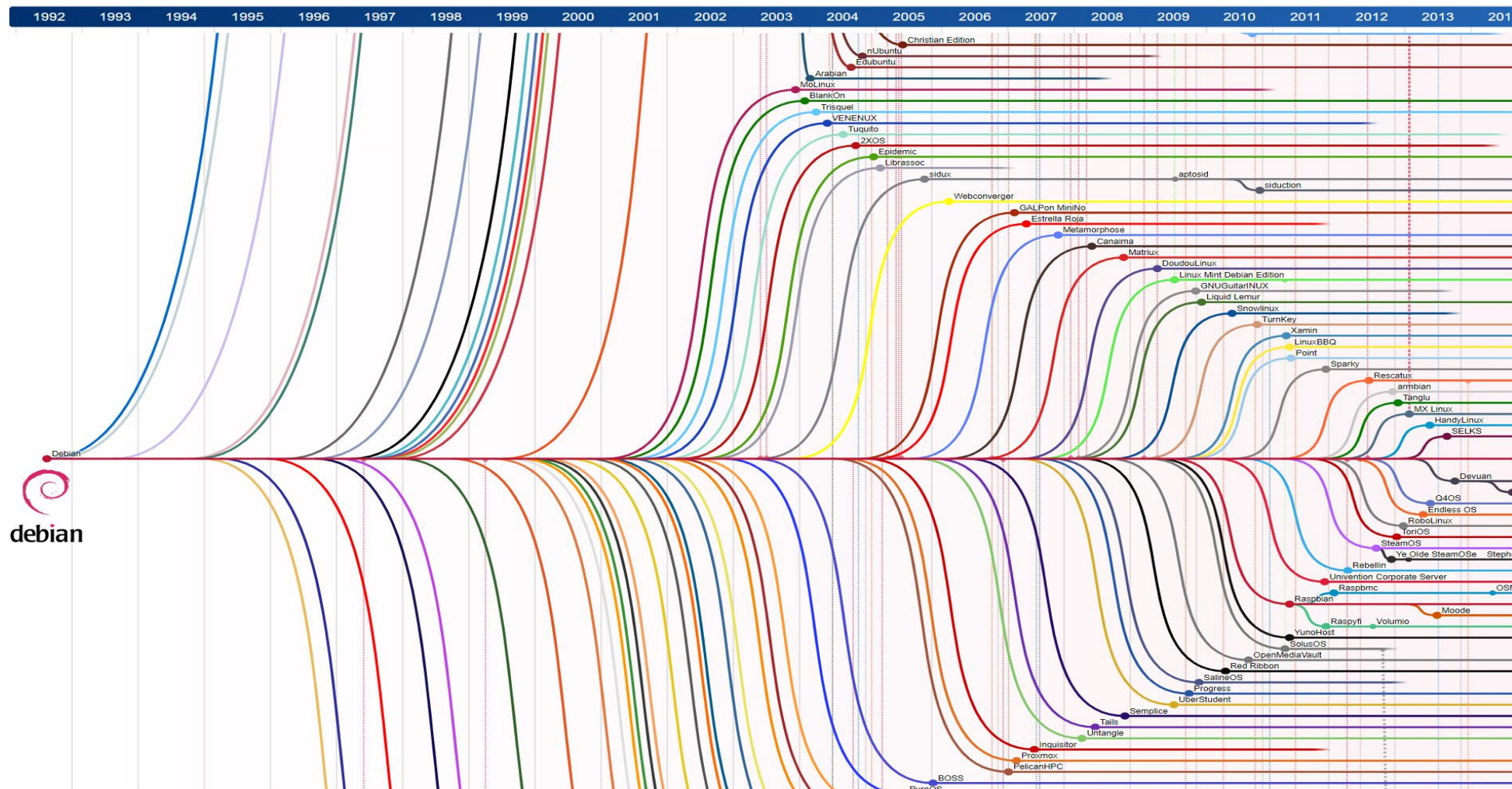
Linux的成功：偶然与必然



Linus Torvalds
(1969-)

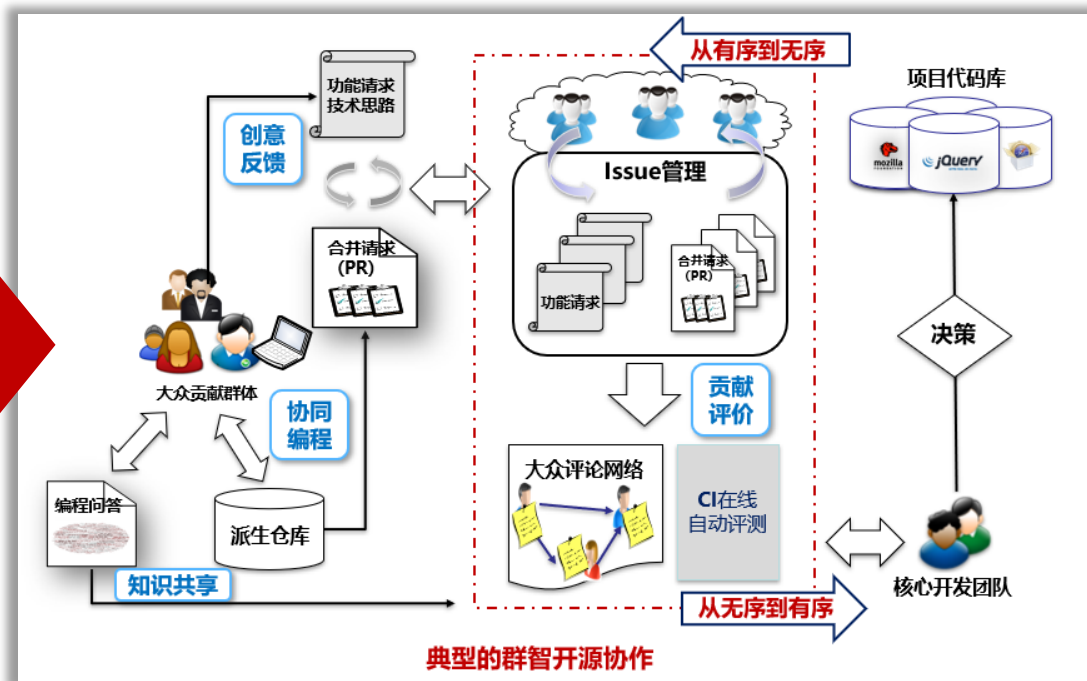
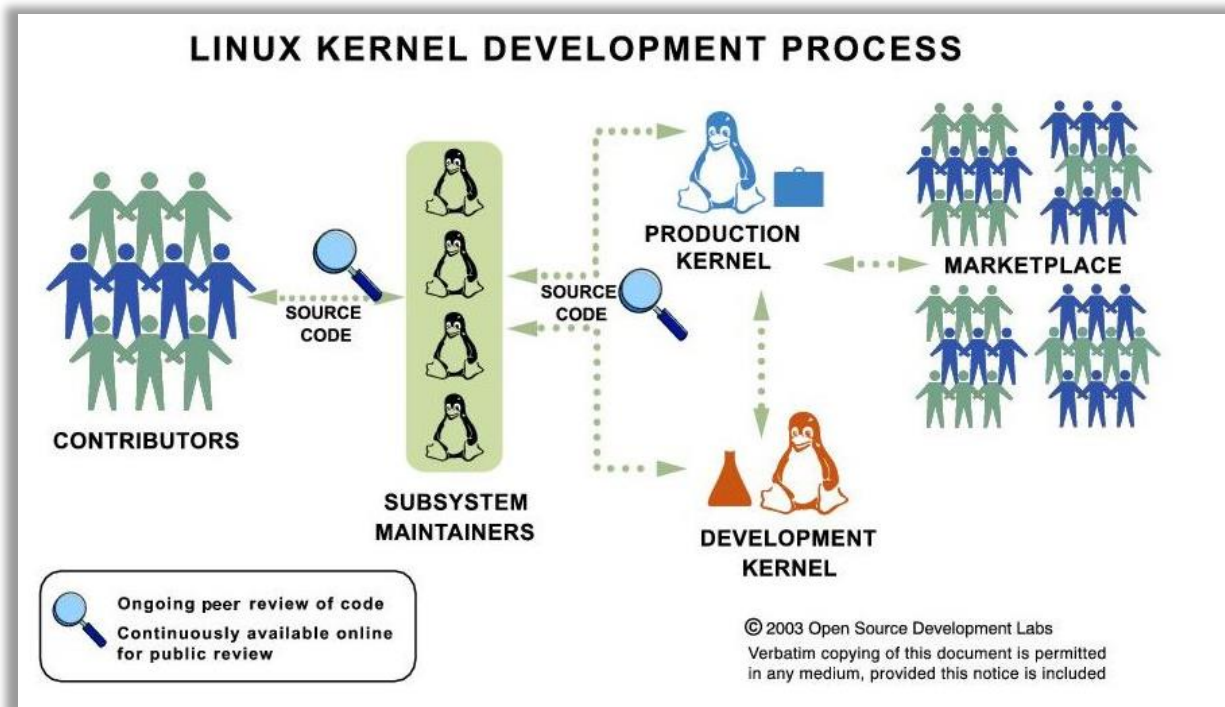


1991年8月发起



Linux的成功：偶然与必然

- 从被组织的大规模协同生产活动到自组织的超大规模协同开发活动
- 第一个大规模松散自由组织的成功案例：超过1900个公司21000个开发者，贡献代码超过2700万行



最新数据（更新至2020年）

https://www.linuxfoundation.org/wp-content/uploads/2020-Linux-Foundation-Annual-Report_120520.pdf

<https://web.archive.org/web/20210315153836/https://data.stackexchange.com/>

开源模式的本质特征

- 用户需求驱动： scratch my own itch
- 开放透明的同行评审： 同行评审的范本
 - Linus law: **given enough eyeballs, all bugs are shallow**
- 人类最佳协作模式：精英自由主义者的一次成功的协作行为艺术的展现（开放自由与规则规范的完美平衡）

大教堂与集市

THE CATHEDRAL & THE BAZAAR

WISDOMS ON LINUX AND OPEN SOURCE BY THE ACCIDENTAL REVOLUTIONARY

开源参与者技能 – 文化和技术



“Publicly making fun of people is half the fun of open source programming.

本质：公开透明同行评审

In fact the main reason to eschew programming in closed environments is that you can't embarrass people in public.”

– Linus Torvalds



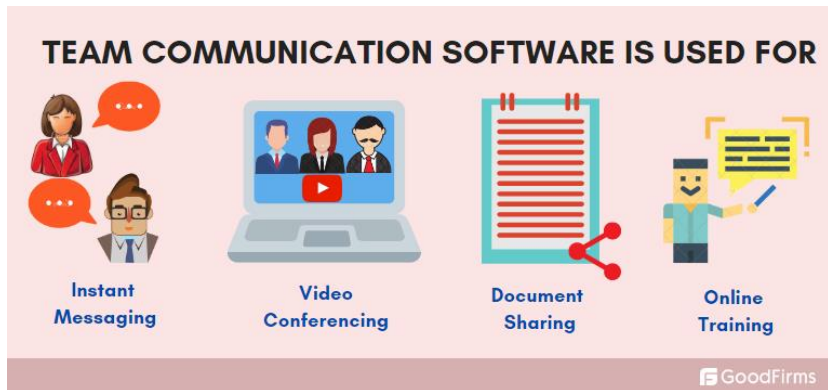
Linux 内核稳定分支(-stable)的维护者

你最喜爱的软件开发工具是什么？你通常使用什么？你在桌面一般运行哪些程序？

我重度依赖 Email 客户端 mutt 和 编辑器 vim，它们是我生存的必需品。其他我在日常工作中会用到的工具有：内核开发中使用 git 和 quilt；浏览器用 Chrome 和 Firefox；irc 通讯使用 irssi；桌面环境选择 GNOME 3，有时候用烦了也会退回到 OpenBox 或 i3m；每过一段时间我也会测试下 KDE，仅仅为了确保能够跟进开发进展。

开源参与者技能 – 沟通和协作

全球分布式开发，沟通工具灵活多样

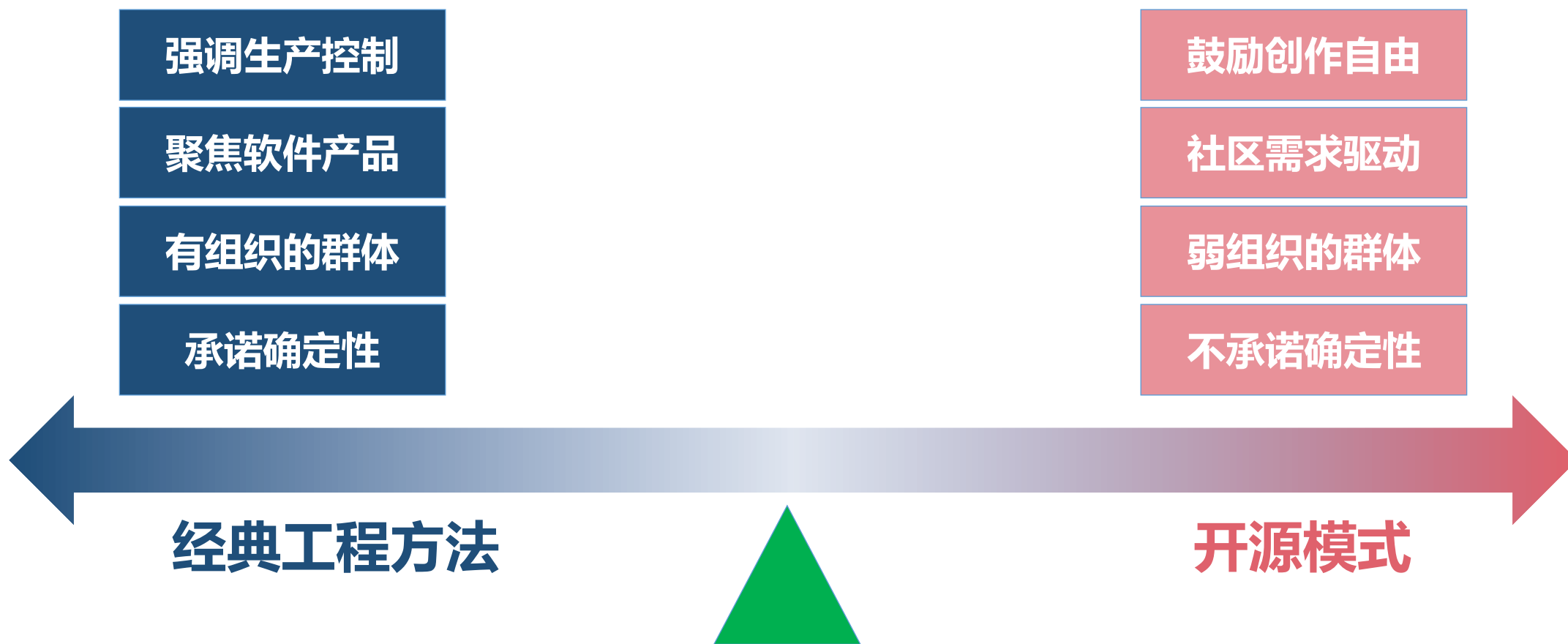


- 软件开发的复杂性： $n*(n-1)$ 的沟通路径， n 是开发者数量
- 开源开发：小内核，驱动大外围.....

开源模式：自下而上、开放协作

元素		传统工程化方法 自上而下，逐步求精	开源模式 自下而上，开放(黑客)合作
理念	需求	有明确用户和明确一致用户需求描述，这是软件开发的前提	没有明确的需求，开发者自我驱动开发软件特征
	质量	源代码满足需求描述的程度	黑客的自我要求及社区口碑
	效率	开发满足需求软件的时间和成本	行业最高水准的人们的自我驱动的效率
方法	过程模型	瀑布模型、敏捷模型、能力成熟度模型	开发者社区的自组织模式
	支持工具	软件产品线工具集	版本控制工具、缺陷追踪工具
	计算环境	面向计算机环境	面向互联网环境

传统工程化方法 vs. 开源模式



开源模式将能够做到的软件工程化以一种最佳模式呈现了出来。

公地悲剧



- 亚里斯多德：“那由最大人数所共享的事物，却只得到最少的照顾。”
- 公共草地上有一群牧羊人，每一个牧羊人都想要多获利一些，所以某个牧羊人就带了大量的羊来放牧，虽然他知道过度放牧，草地可能会承受不住。但他依然获利了，而后所有的牧羊人都跟进，所以草地牧草耗竭，悲剧因而发生了。
- 这个案例揭示有限的资源注定因自由使用和不受限的要求而被过度剥削。这样的情况之所以会发生源自于每一个个体都企求扩大自身可使用的资源，然而资源耗损的代价却转嫁所有可使用资源的人们。（可使用资源的群体数目可能远大于夺取资源的数目）

开源软件开发会发生公地悲剧吗？

- 开源软件是公共财产
- 开源软件的悲剧：
 - Linus Law: 足够多的眼睛，就可让所有问题浮现。
 - Heartbleed – OpenSSL:
<https://www.technologyreview.com/2021/12/17/1042692/log4j-internet-open-source-hacking/>

Project history [\[edit\]](#)

The OpenSSL project was founded in 1998 to provide a free set of encryption tools for the code used on the Internet. It is based on a fork of [SSL](#) by Eric Andrew Young and Tim Hudson, which unofficially ended development on December 17, 1998, when Young and Hudson both went to work for [RSA Security](#). The initial founding members were Mark Cox, Ralf Engelschall, Stephen Henson, [Ben Laurie](#), and Paul Sutton.^[3]

As of May 2019,^[4] the OpenSSL management committee consisted of 7 people^[5] and there are 17 developers^[6] with commit access (many of whom are also part of the OpenSSL management committee). There are only two full-time employees (fellows) and the remainder are volunteers.

The project has a budget of less than one million USD per year and relies primarily on donations. Development of TLS 1.3 is sponsored by Akamai.^[7]



Alerts and Tips Resources

National Cyber Awareness System > Alerts > OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160)

Alert (TA14-098A)

OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160)

Original release date: April 08, 2014 | Last revised: October 05, 2016

[Like](#) [Print](#) [Twitter](#) [Facebook](#) [LinkedIn](#) [Share](#)

Systems Affected

- OpenSSL 1.0.1 through 1.0.1f
- OpenSSL 1.0.2-beta

Overview

A vulnerability in OpenSSL could allow a remote attacker to expose sensitive data, possibly including user authentication credentials and secret keys, through incorrect memory handling in the TLS heartbeat extension.

Description

OpenSSL versions 1.0.1 through 1.0.1f contain a flaw in its implementation of the TLS/DTLS heartbeat functionality. This flaw allows an attacker to retrieve private memory of an application that uses the vulnerable OpenSSL library in chunks of 64k at a time. Note that an attacker can repeatedly leverage the vulnerability to retrieve as many 64k chunks of memory as are necessary to retrieve the intended secrets. The sensitive information that may be retrieved using this vulnerability include:

- Primary key material (secret keys)
- Secondary key material (user names and passwords used by vulnerable services)
- Protected content (sensitive data used by vulnerable services)
- Collateral (memory addresses and content that can be leveraged to bypass exploit mitigations)

Exploit code is publicly available for this vulnerability. Additional details may be found in CERT/CC Vulnerability Note VU#720951.

开源软件开发跟公地悲剧的区别

- 从使用上来说：开源软件的广泛使用呈现网络效应，不会因使用而发生损耗和枯竭。
- 从供应上来说：开源软件的供应如何解决？如何保障无偿开发的人们动力？

经典开发过程和开源开发过程的 实际案例？

Call for contribution: 企业内部发布一个产品版本的全过程, 和开源社区发布一个产品版本的全过程

-- 下述片子来自: Kata container王旭

开源开发：开放性带来的层级提升



你修了一座独木桥用来过河

比较一下
妈妈已经不认识了



后来又有人扩展出来了过火车的功能



来了一个家里有车的家伙，改了改



又有个家伙为了保证过船，把桥抬高了

这样的事情，真真切切地发生过.....

以上图片来自Google Images搜索

Kubernetes & the Runtimes




- Kubernetes是用于自动部署、扩展和管理“容器化（containerized）应用程序”的开源系统。
- 起初，Kubernetes只支持Docker，使用Docker来创建容器
- 后来，2015年，CoreOS把rkt加入Kubernetes，Hyper尝试加入runV，但是...太乱了，于是...
- 2016年，大家一起合写了CRI(运行时接口)，允许了不同Runtime
- 2017年，Kata Containers发布，随后2018年gVisor发布，兼容docker containerd 太繁琐了，于是.....
- 2018年Google推动了containerd shim-v2，得到大家支持，而且
- Kubernetes添加了RuntimeClass，用户可以指示使用什么Runtime
- 2020年，出现了CVE-2020-14386容器逃逸漏洞，Google表示使用gVisor的GKE用户不受此漏洞影响（:Google Kubernetes Engine是一个Kubernetes管理平台，主要在谷歌云平台上运行）
- Kubernetes与安全容器运行时这件事情，其实完全不在初始设计之内，是自治的开源社区的演进的结果

Kubernetes vs. Docker

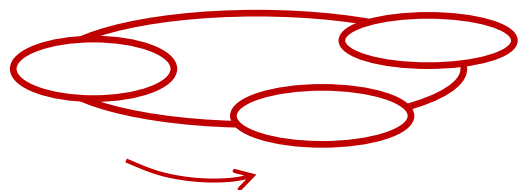
- While Docker is a container runtime, Kubernetes is a platform for running and managing containers from many container runtimes.
- Kubernetes supports numerous container runtimes including Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

Kubernetes VS Docker

Comparison Chart

Kubernetes	Docker
Kubernetes is an ecosystem for managing a cluster of Docker containers known as Pods.	Docker is a container platform for building, configuring and distributing Docker containers.
Kubernetes is not a complete solution and uses custom plugins to extend its functionality.	Docker uses its very own native clustering solution for Docker containers called Docker Swarm.
Load balancing comes out of the box in Kubernetes because of its architecture and it's very convenient	The load balancer is deployed on its own single node swarm when pods in the container are defined as service.
Takes relatively more time for installation.	Setup is quick and easy compared to Kubernetes.
	

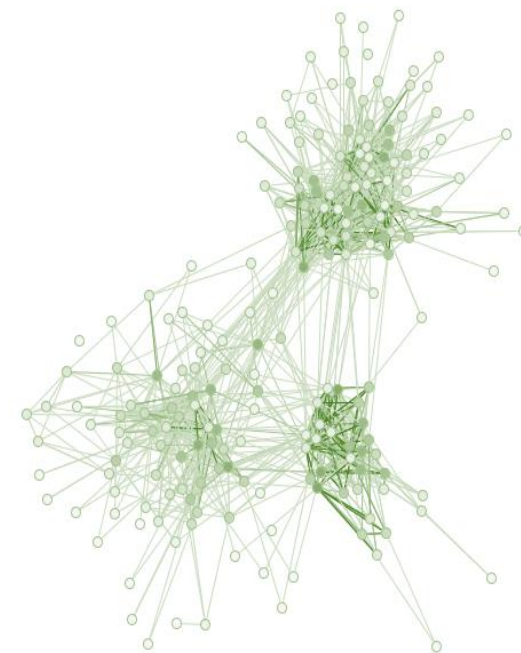
开源开发的核心挑战



群体如何激发？（人们为什么无偿做贡献？）

群体如何组织？（当企业加入，开源模式是否变成传统工程化模式？）

群智平台与生态？（如何提供自动化技术和工具支持群体开发、如何形成生态？）



鸣谢
王涛
王旭