Q1

Linus torvalds 说过：never break user space，对开发者来说意味着：我们必须保证在引入新代码时，老代码仍然能正常工作，Linux只会保留一个越来越小的核心内核，其他所有内核功能都由用户定义。Kubernetes 和 docker 是go语言开发、体现这种趋势的2个著名开源项目。

**<https://github.com/docker>**

**https://github.com/kubernetes/kubernetes**

k8s（Kubernetes）能处理的节点和群集大小非常灵活和自动化，很多互联网公司使用它。 但是2021.1月据统计，所有K8s集群的80%观察到的节点少于10个，低于5个节点的占60％。K8s可以大规模扩展，大多数人没有那样使用它。



同时：虽然 Kubernetes 总体上极受欢迎，但规模最小的公司（1-5 名员工）更常 使用 Docker Swarm（41% 的公司使用 Swarm，31% 的公司使用 Kubernetes）

请你描述下软件维护活动的总工作量的公式，并且用公式的P,K,C,D各部分分析一下出现这种现象的原因。

参考答案：

1. 软件维护活动分为生产性活动和非生产性活动。生产性活动包括分析评价、修改设计和编写程序代码等；非生产性活动包括理解程序代码功能、数据结构、接口特点和设计约束等。维护活动的总工作量可以用以下公式表示：

　　　　　　　　　　M＝P＋K×exp（C－D）

其中，M表示维护工作的总工作量，P表示生产性活动的工作量，K表示经验常数， C表示复杂性程度，D表示维护人员对软件的熟悉程度

2. P 是互联网公司的运维或者开发人员了解k8s/docker swarm的架构和功能，理解它们应用场景、安装、运行甚至进行fork和fix修改的工作量， D是运维和开发人员对k8s/docker swarm的熟悉程度（个人开发者和小公司往往对docker更为熟悉，而大公司因为业务需要，有大规模集群需求，所以往往倾向于k8s)， 复杂性C上docker比k8s复杂性低很多，K作为经验常数，普通开发者和运维人员的docker的M值要大于k8s.

Docker Swarm 适用于简单和快速开发至关重要的环境，而 Kubernetes 适合大中型集群运行复杂应用程序的环境，所以造成上面的现象。

Q2

霍尔逻辑发明人霍尔说：“世上有两种设计软件的方法。一种是尽量的简化，以至于明显没有任何缺陷。而另一种是尽量复杂化，以至于找不到明显的缺陷。”软件架构模式的发展和大大简化了程序的开发流程。

除了软件架构模式的发展之外，我们还看到 Docker 和 Kubernetes 等新技术的出现，用于支持我们的软件基础设施，实现对可伸缩产品和服务的高效管理。我们已经从硬件虚拟化发展到容器化。

请你从软件维护为什么困难的角度，结合docker等容器化技术流行的原因，重点描述容器化技术对软件部署维护的改进？

参考答案：

软件维护是一件十分困难的工作，其原因主要是由于软件需求分析和开发方法的缺陷造成的。软件开发过程中没有严格而又科学的管理和规划，便会引起软件运行时的维护困难。软件维护的困难主要表现在以下几个方面：

　　（1） 读懂别人的程序是很困难的，而文档的不足更增加了这种难度。一般开发人员都有这样的体会，修改别人的程序还不如自己重新编写程序。

大多数业务应用程序由几个单独的组件组成，这些组件组织成一个堆栈- 一个Web服务器，一个数据库，一个内存缓存。container可以将这些部件组合成具有易于更换的部件的功能单元，不再关心内部程序，降低读懂的要求。每件作品都由不同的容器提供，可以独立于其他容器进行维护，更新，换出和修改。

　　（2） 文档的不一致性是软件维护困难的又一个因素，主要表现在各种文档之间的不一致以及文档与程序之间的不一致性，从而导致维护人员不知所措，不知怎样进行修改。这种不一致性是由于开发过程中文档管理不严造成的，开发中经常会出现修改程序而忘了修改相关的文档，或者某一个文档修改了，却没有修改与之相关的其他文档等现象，container提供了一种高效和高粒度的机制，用于将软件组件组合到现代企业所需的各种应用程序和服务堆栈中，并保持这些软件组件的更新和维护，并且相关的文档中很大一部分被“程序化”为配置文件和固化的部署脚本，降低了对文档的需求。

　　（3） 软件开发和软件维护在人员和时间上存在差异。如果软件维护工作是由该软件的开发人员完成，则维护工作相对比较容易，因为这些人员熟悉软件的功能和结构等。但是，通常开发人员和维护人员是不同的，况且维护阶段持续时间很长，可能是10～20年的时间，原来的开发工具、方法和技术与当前有很大的差异，这也造成了维护的困难。container技术则让开发人员维护容器，运维和开发人员变得更紧密。

　　（4） 软件维护不是一件吸引人的工作。由于维护工作的困难性，维护经常遭受挫折，而且很难出成果，所以高水平的程序员自然不愿主动去做，而公司也舍不得让高水平的程序员去做。docker直接让程序员自己可以维护软件部署环境，间接提升了软件维护人员的技术水平，提供了维护质量。

Q3

伯克利科学家Steven Weber说：“软件或许就是推动当今技术革命的蒸汽机，而开源社区的想法比起所产生的代码更具有深远意义。开源已经改变，并将继续改变软件开发的方式”

git 是开源社区标配软件，git工作流程中的fork工作流程如下：



假设你公司发现k8s在95%时候满足要求，但是有需要公司开发-运维团队根据开源项目文档进行一些接口函数修改，以便于部署到一种新的硬件系统。这种行为属于Reverse Engineering还是Re-engineering？

Reverse Engineering和Re-engineering在软件工程阶段时间上有啥区别？

你知道一些代码维护的技巧，方便你的PR快速被其他项目维护者接受吗？

参考答案

1.

属于Re-engineering， 再工程不仅能从已有的程序中重新获得设计信息，而且还能使用这些信息改建或重构现有的系统。

Reverse Engineering用于软件的起始建造阶段，而Re-engineering用于软件后续的修改阶段。

2.

使用小的提交，将不同功能的代码分拆到不同的提交甚至是不同的 Pull Request 中

必要的逻辑添加注释说明变更的理由

遵循代码约定，如 [Coding Conventions](https://github.com/kubernetes/community/blob/master/contributors/guide/coding-conventions.md)、[API Conventions](https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-conventions.md) 和 [kubectl Conventions](https://github.com/kubernetes/community/blob/master/contributors/devel/sig-cli/kubectl-conventions.md)

确保修改部分可以本地跑过单元测试和功能测试

使用 [Bot 命令](https://prow.k8s.io/command-help) 设置正确的标签或重试失败的测试

Q4

《大教堂和集市》的作者曾言：给与足够的眼球，所有的Bugs都很容易发现（例如，大量的beta测试，结对开发，所有的问题都能很快的发现和修复）。但是事情并不只是发现bug那么简单，请简要说明通过修改代码修复的副作用。

我发现一个好用的开源的docker image，但是它的部署脚本：docker file刚好差一个组件，我需要对docker file进行组件添加，才能部署到我希望部署的特定硬件平台。这种行为对软件的可维护性有什么影响？它属于改正性维护还是适应性维护？

如果一定要通过修改代码的方式做软件维护，推荐什么顺序进行修改？

怎么对开发中关于维护进行合理的估计？

参考答案：

0.增加了组件，相当于增加了复杂度，应该会降低系统可维护性。

1.首先，当一个错误被更正时，修正本身可能带来新的系统问题。其次，进行修正时，系统结构发生了相应的变化。因为许多修理是在有限的目标下来解决特定的问题的，相关的组件，象面向对象系统中的继承关系那样， 也常常退化。

2.- 确定改动点：画草图、影响结构图。

- 找到测试点 ：在影响结构图根部

- 解依赖：各种解依赖技术、解耦合的技术对代码进行修改

- 编写测试：TDD（最好有编写测试）

* 修改、重构：特征图

3. 开发中关于维护的估计。特别是，最好的估计是基于过去的对于相似的 工程的整个历史的。另外，当工程或者产品的属性发生改变时，这个预测必须重新进行;既 然遗留下来的系统不断地改变，这个估计也基于定期的改变。

Q5

《计算机程序的构造和解释》（简称为SICP）的作者说过：“代码是写给人看的，不是写给机器看的，只是顺便计算机可以执行而已”

一个软件工程要真正长期可用，必须具备可维护性和可阅读性。

你和同事一起开发一个项目，各自本地同时维护了一组服务器的docker-file 和部署的bash脚本，怎么度量双方代码的可维护性优劣？

从软件可维护性指数角度回答，使用dockfile 相比较以前单纯本地手工安装依赖方式的好处在哪里？

参考答案：

1.

圈复杂度（Cyclomatic Complexity）：度量代码的结构复杂度。

代码行数：……

可维护性指数（Maintainnbility Index）MI：计算0到100之间的索引值。表示维护代码的相对容易性，高价值意味着更好地可维护性。

继承的层次数（Depth of Inheritance）：就是继承深度，英文更通俗些。

类之间的耦合度（Class Coupling）：通过参数，局部变量，返回类型，方法调用，泛型或模板实例化，基类，接口实现，在外部类型上定义的字段和属性修饰来测量耦合到唯一类。

单元测试覆盖率（Unit test coverage）

2.

从可维护性角度来说提升了可维护性指数：

Dockerfile解放了手工操作很多步骤  
Dockerfile保证了环境的统一

使用dockerfile使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署该镜像