《开源软件开发技术》2020年作业（2）

班级 大数据1901 学号 2019011819 姓名 邢云帆

1. 开源开发有哪些人员角色？

****维护者****

****贡献者****

****修订者****

****对于某些项目来说， “维护者”**** 就是唯一拥有提交权限的人。

作为一名维护者，不一定非得一定要为项目撰写代码。Ta有可能是项目的布道师，为项目的宣传做了很多的工作，又或者是撰写文档让更多的人参与进来。不管他们每天做什么，维护者就是那些对项目方向负责的人，并致力于项目的改进。

****作为 “贡献者” 可以是任何人**** ，只要Ta提出issue或PR 就叫做贡献者，那些为项目作出有价值的都算（无论是分类问题，编写代码还是组织会议），又或者是将他们的PR合并进主干的（或许这个定义是最接近所谓的贡献者的）。

****“修订者”**** 可能用于区分其他形式的贡献的提交访问，这是一种特定类型的责任。

其实你可以根据自己喜欢的方式来定义项目的角色，[考虑使用更广泛的定义](https://www.w3cschool.cn/opensourceguide/opensourceguide-prkf3bpr.html" \t "https://www.w3cschool.cn/opensourceguide/_blank) 来鼓励更多的形式的贡献。无论技术技能如何，您都可以使用领导角色来正式识别为您的项目做出突出贡献的人员。

2. 创建和开展开源项目需要注意的问题有哪些？

如果你想发布一个开源库，请确保它有以下特点：

清晰的依赖性和安装说明

至少有一个简要的文档指南

修改日志和仓库中的标签

关于支持的语言、运行时、工具版本的信息和项目的成熟度

一个可以让用户提问和交流的邮件列表

缺少任何一项都会造成一些用户的愤怒和沮丧，当然同时也浪费了时间。

### **怎样让你的开源项目更棒**

每年，越来越多的人发布了自己开发的库并且它们开源。这里我们分享一些我们经验，以便你的用户对你的库满意。

### **创建一个实用的README**

即使你的项目有一个很棒的网站，潜在的用户第一次接触这个项目很可能就是通过阅读README文件。我们需要确保它很棒并且包含了有用的信息。

### **提供依赖信息**

那么说你会发布你的开源项目。这说明你很聪明，真有你的！不幸的是，不是所有人都像你那样，而且有一些人对这门语言或者你在做的系统完全不了解。这意味着对你来说很显然的事情对他们来说就一点也不显然了。

其中一件就是缺少依赖说明或者安装说明

我到底怎么安装这个东西，难道不能说得清楚一些吗?

这很快就能让用户生气。在源代码里找你的包或者构件的名字是很烦人的，有些项目对构件使用一些特别有才的名字，完全不符合仓库的名字。

让你的用户从这些糟糕的事情中脱离出来吧。问题是怎样添加依赖性，理想状况下可以通过复制粘贴一小段代码。

如果需要例子的话可以点击 [Welle](https://github.com/michaelklishin/welle" \l "maven-artifacts)。

### **清楚的说明项目的成熟度**

你在生产中使用这个项目有几个月了吗？你是否觉得它还是不完整的？你是否希望API在下一个版本会彻底地修改？你的项目是否在要求最多并且很老的项目中也能稳定安全的使用？

要把这些说得清楚。下次你就不会因为做了一个错误的介绍，但是没有的提供任何项目成熟度的信息而项目浪费一周的时间了。你会意识到几句短短的话就能产生很大的影响。

### **运行时、语言、工具版本的文档支持**

当考虑到向后兼容时，Clojure有一个很好的跟踪记录。它好的几乎让人难以置信。包括1.2到1.3的升级，之后的升级对绝大多数的项目来说就是一个简单替换。同样地，那些高于1.2的项目大多使用了最新的稳定版本。

然而，不会一直都是这样。在某些情况下，未来版本的Clojure会打破兼容性。我们怎么让我们的用户不愤怒？通过在README中清楚的说明哪些版本是支持的。

这只需要写一行文字。这样，在你发布的那一周就少了抱怨，同时也减少了初学者的很多麻烦。

如果你需要一个例子，有一个来自 [Welle的例子](https://github.com/michaelklishin/welle" \l "supported-clojure-versions)。

### **说明你使用了什么许可证**

你可能并不太关心许可证，但是那些在大公司中想用你的库的人很关心。他们必须知道！当他们想用Clojure/[Node.js](http://blog.jobbole.com/1254/" \o "Node.js 究竟是什么？" \t "https://www.oschina.net/news/48233/_blank)/Scala/Go等等的时候，可能不能使用。

因此清楚的说明你的许可证。也请你使用一些对商业友好的协议，除非你有自己的理由。（ [Apache Public License 2.0](http://www.tldrlegal.com/license/apache-license-2.0-(apache-2.0)和[Eclipse Public License](http://www.tldrlegal.com/license/eclipse-public-license-1.0-(epl-1.0)）是不错的选择。注意到一些许可证(比如[MIT](http://www.tldrlegal.com/license/mit-license))的确很友好、流行，但是不提供任何专利保护，在当前的法律环境下也不应该忽视。

最后，记得你可以使用双许可证，如果你真的是许可证中立的话可以使用，比如APL2/GPLv2。那个你的用户就可以选择最适合他们的许可证了。

疑惑的时候，可以参考[摘要：合法、开源许可证用白话概括](http://www.tldrlegal.com/)（但是别把它当作合法的建议）

### **为你的项目写文档**

写文档不容易同时也是需要花费一些时间的。然而，文档是你能为你的用户做的最好的事了。不仅能够节省他们大量的时间，也可以让他们确信你的库不是被遗弃的软件。

文档能够让你的用户完成他们起初使用你的库的任务。像Rob Pike说的，它“让这些任务成为可能”。这让你的用户知道你重视这一点，让他们知道你是个有血有肉的人，不是一个产生代码的机器。

更好的是，如果你开源了你的文档站点(这似乎没有理由不那么做)，你会看到人们会比贡献代码的修改更早的贡献出小的改进。

### **更容易升级**

某些时候，你想要发行项目的另一个版本。这可能是让你的用户很开心，因为他们已经使用了你的库，或者很生气，浪费了他们时间。

### **不关心向后兼容**

关于软件开发的一件很令人生气的事就是当你升级一个库但是数百个测试失败了。更让我生气的就是我还要重写我一半的基础代码，因为有人在没有任何警告的前提下决定打破公共的API。

因此，致力于维护向后兼容性。当然你没有必要像OpenJDK那样支持15年以前的项目。但是在移除之前建议不使用一些东西能够更容易发现哪些地方改动了。

你怎么做到这点呢？维护一个修改日志。

### **拥有一个修改日志**

所有这些问题都能通过一个修改日志得到解答。它像推特一样只不过它真的很实用，它是这样用的：

每次你解决一个bug，在日志里加一个简单的记录

每次你加入一个新特性，在日志里简单地提一下，并且用几个代码例子解释它。

每次你做了重大的API改动，在日志中用****粗体****清楚的说明

就是这些了。没有第三步！

### **给版本加上标签**

又是那个时候了，你已经升级版本并且马上就要发布构件了。停一停，先做一件事：给这次提交加上标签。没有标签的话，找两个版本之间的不同会很痛苦的。

一些依赖性(比如Bundler, Rebar)和配置管理工具可以使用标签，开发者系统这些标签是可用的。

使用统一的版本信息，比如v1.0.0-alpha1, v1.0.0, v1.1.2等。标签不一致绝对会导致运维的人整天讨厌你的项目。

### **宣布版本发行**

在你发布一个版本字之后就是要写一个博客日志，或者在你们项目的邮件列表或更大的相关的邮件列表中发个更新(比如[Clojure邮件列表](https://groups.google.com/group/clojure)或者[RabbitMQ](https://lists.rabbitmq.com/cgi-bin/mailman/listinfo/rabbitmq-discuss))

确保主题是以ANN或者[ANN]开头的，这意味着这是一个通告。比如

ANN Welle 1.5.0 发布了

在你的通告中，清楚的说明你的项目是做什么的，它是否向后兼容，并且有到修改日志的链接，可以让用户找到更多的细节。就是这样了。

3. 开源开发为什么要进行版本控制，如何进行版本控制？

如果说什么是软件开发项目一定要使用的基础工具，那么版本控制系统应该算最重要的部分。不管是个人开发或是团队协作开发，都可以通过版本控制系统获得巨大的好处。

　　没有版本控制系统的话，代码可能被别人或自己不小心覆盖或遗失、也不知道是谁因为什么原因改了这段代码、也没办法可以复原回前几天的修改。有了版本控制系统，开发人员只要将每次程式码的变更都纪录（Commit）起来，并且透过版本控制系统中进行更新。

　　有了版本控制系统，我们可以浏览所有开发的历史纪录，掌握团队的开发进度，而且作任何修改都不再害怕，因为你可以轻易的复原回之前正常的版本。我们也可以透过分支和标签的功能来进行软件发行的不同版本，例如稳定版本、维护版本和开发中版本。

很多项目需求方还没有明白开发的定义，这里必须要跟大家说一点老生常谈的段子：“开发永远是个过程，而不是结果。”所以开发者一定要使用版本控制系统，Git或Mercurial是免费开源的版本系统系统、随处可用的网络、便宜的云端服务器，甚至有现成的第三方服务Github。

接下来是几点使用版本控制系统的建议：

1）.将所有东西都放进版本控制系统

　　2）.频繁且适当大小的递交

3）.良好的递交信息

　4. 注册Github网站，填写个人资料（个人介绍，位置，个人网站等）。在Github创建一个仓库（repository），把本作业的文档上传到这个仓库中。将你的Github地址写到下面作为答案供检查。

https://github.com/EternalTwilight