

06 | 大厂都在用哪些敏捷方法？（上）

2019-03-07 宝玉 来自北京

《软件工程之美》



你好，我是宝玉，我今天分享的主题是：大厂都在用哪些敏捷方法？我将分为上下两篇，来与你一起讨论这个话题。

在我还是一个野路子程序员，到处接私活做网站时，就开始好奇：大厂都是怎么开发软件项目的？直到毕业后，我前前后后加入了若干大中小型企业，包括这些年在美国高校、公司的一些经历，对大厂的项目开发有了比较多的了解。

其实大厂做项目也没有什么特别的，无非就是工程中常见的“分而治之”的策略：**大项目拆成小项目，大服务拆成小服务，大团队拆成小团队。**

服务之间通过商定好的标准协议进行通信，架构上将大的服务拆分离成微服务，大团队按照业务或者服务拆分成小组，按照一定的流程规范保障协作。最终，各个小组要负责的内容其实就不多了。

就像淘宝这种网站，不需要一个庞大的项目组，通过逐级分拆，一个小组可能就只需要负责一个页面中的一个小模块。

所以，也要归功于现在微服务、容器等新技术，可以将复杂的业务逐级拆分，让很多公司能真正敏捷起来。

在上一篇文章中，我有提到，团队要实施敏捷，不仅要小，还要组织扁平化。相对来说，美国的互联网大企业做的还是很不错的，组织架构都很扁平，工程师地位很高。

这些年，国内工程师地位应该也有很大提升，组织也在向扁平化发展。前些天我也看到阿里工程师写的一篇文章《[敏捷开发的根本矛盾是什么？从业十余年的工程师在思考](#)》，对这个问题有精彩的论述。

下面，我就带你一起看看，大厂具体是怎么应用敏捷方法的。

和敏捷开发相关的主要流程规范

大厂里流程规范很多，最开始你会不喜欢它们，后来会离不开它们。

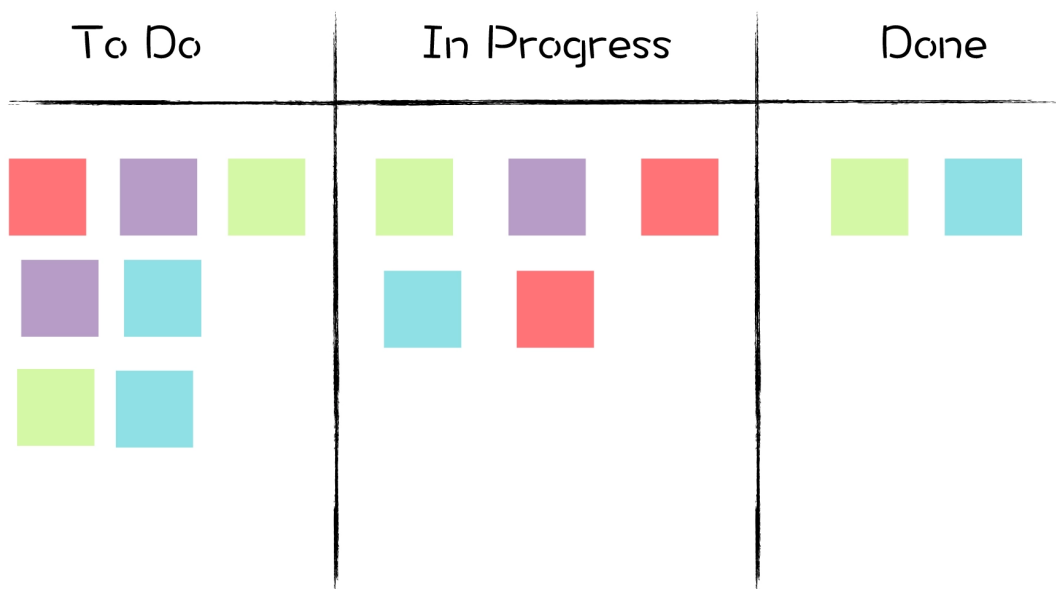
这些墙很有趣。刚入狱的时候，你痛恨周围的高墙；慢慢地，你习惯了生活在其中；最终你会发现自己不得不依靠它而生存。这就叫体制化。——《肖申克的救赎》

这里，我简单将其中和敏捷开发相关的流程介绍一下。

一切工作任务围绕 Ticket 开展

早些年的项目开发，都是围绕着项目计划开展的，把甘特图打印贴在墙上，方便团队成员看项目进展到什么地步了。自从敏捷化后，开始变成了看板。

所谓的看板，就是把白板分成几个栏，每一栏为一类，分别写着“To Do（待选取）”、“In Progress（进行中）”、“Done（完成）”等，再把工作任务变成一个个五颜六色的即时贴，根据状态贴在不同的栏下面。



慢慢的物理的看板变成了电子看板，通过各种项目管理软件来管理跟踪这些任务，即时贴也变成了 Ticket（也有叫 Issue 的）。逐渐的，所有与开发相关的任务也都和 Ticket 挂钩了：

报一个 Bug，提交一个 Ticket ；

提一条需求，提交一个 Ticket ；

要重构一下代码，提交一个 Ticket 。

看板这种基于 Ticket 来管理跟踪任务的方式，看起来繁琐，但确实是很高效的一种方式。

每一个任务的状态都可以被跟踪起来：什么时候开始做的，谁在做，做完没有。

整个团队在做什么一目了然。

Ticket 和敏捷开发中的 Backlog（任务清单）正好结合起来，通过 Ticket 可以收集管理整个项目的 Backlog 和当前 Sprint（迭代）的 Backlog。

有了看板后，大家每天上班第一件事就是打开看板，看看当前 Sprint 还有哪些 Ticket 没有完成，哪些已经完成，哪些正在进行中，非常直观。

作为项目成员来说，做完手头的事情也不用去问项目经理该干什么事情了，直接从 To Do 栏选一条 Ticket 做就是了；对于项目经理，看看 To Do 栏还有多少没有被选取，就知道还剩多

少 Ticket 没完成，看看 In Progress 栏就知道哪些 Ticket 正在进行中。

如果有 Ticket 在这一栏待太久或者这一栏 Ticket 太多，那可能就有风险了，就可以及时介入。

对于项目管理软件和 Ticket，我在后面章节中还会有进一步介绍。

基于 Git 和 CI 的开发流程

如果你的团队应用瀑布模型来开发，大概会有两大烦恼：**代码不稳定和部署太麻烦。**

早些年虽然也用源代码管理，但是大家都是在 master（主干）上开发的，所以 master 的代码特别不稳定，一不小心就可能被人签入了不稳定的代码。所以在上线前，有一段时间叫“代码冻结期”，意思就是这期间，除非是紧急修复，否则谁都不能往上面提交代码。

还有，测试环境的部署也是个老大难问题，尤其是在服务较多时，编译要注意各种依赖和环境的配置。所以更新测试环境是个大工程，以至于当年我在飞信的时候，专门有人负责部署测试环境。

上面的“代码冻结”和“专人部署”方案，可一点都不敏捷。所以团队想要敏捷起来，一定要解决代码不稳定和部署太麻烦这两个大问题。

好在基于 Git 的开发流程结合 CI 的自动测试部署，很完美的解决了这两大问题。

Git 本来只是源代码管理工具，但是其强大的分支管理和灵活的权限控制，结合一定的开发流程，却可以帮助你很好的控制代码质量。

我们假设现在 master 的代码是稳定的，那么怎么保证新加入的代码也稳定呢？

答案就是代码审查（Code Review）和自动化测试。如果代码有严格的审查，并且所有自动化测试代码都能测试通过，那么可以认为代码质量是可靠的。当然前提是自动化测试代码要有一定的覆盖比率。

关于这点，对于大厂来说倒不是什么问题，正规的项目组对于代码审查和自动测试代码的覆盖率都有严格的要求。现在还有一个问题，就是如何在合并到 master 之前把代码审查和自动化测试做好呢？

简单来说，就是每次要往 master 添加内容，不是直接提交代码到 master，而是先基于当前稳定的 master，克隆一个 branch（分支）出来，基于 branch 去开发，开发完成后提交一个 PR（Pull Request，合并请求）。

Merged

inline values as decorators when debugging #16129

Changes from all commits

File filter...


Jump to...

Diff settings


Review changes

166 src/vs/workbench/parts/debug/electron-browser/debugInlineDecorators.ts Show comments Copy path View file


@@ -0,0 +1,166 @@
1 + /*-----
2 + * Copyright (c) Microsoft Corporation. All rights reserved.
3 + * Licensed under the MIT License. See License.txt in the project root for license information.
4 + *-----*/
5 + 'use strict';
6 +
7 + import { IDictionary } from 'vs/base/common/collections';
8 + import { IDecorationOptions, IRange, IModel } from 'vs/editor/common/editorCommon';
9 + import { StandardTokenType } from 'vs/editor/common/modes';
10 + import { IExpression } from 'vs/workbench/parts/debug/common/debug';
11 +
12 + export const MAX_INLINE_VALUE_LENGTH = 50; // Max string length of each inline 'x = y' string. If exceeded ... is added
13 + export const MAX_INLINE_DECORATOR_LENGTH = 150; // Max string length of each inline decorator when debugging. If exceeded ... is added
14 + export const MAX_NUM_INLINE_VALUES = 100; // JS Global scope can have 700+ entries. We want to limit ourselves for performance
15 + export const MAX_TOKENIZATION_LINE_LEN = 500; // If line is too long, then inline values for the line are skipped
16 + export const ELLIPSES = '...';
17 + // LanguageConfigurationRegistry.getWordDefinition() return regexes that allow spaces and punctuation characters for word tokens.
18 + // Using that approach is not viable so we are using a simple regex to look for word tokens.
19 + export const WORD_REGEX = /[^\s_A-Za-z][^\s_A-Za-z0-9]*/g;

 isidorn on Dec 5, 2016 Contributor


For simplicity can we just use the regex \b word boundary character?

 nojvek on Dec 5, 2016 Author Contributor

\b doesn't work since it doesn't consider \$ and _ as word character. Powershell, php would break seriously. Javascript allows \$ and _ in variable names so they would have quirks as well.

 isidorn on Dec 9, 2016 Contributor

Ok makes sense. Though I hate introducing new regexes. Once we merge this in I can investigate if we can reuse some other word logic we have

 Reply...

Start a new conversation

20 +
21 + export function getNameValueMapFromScopeChildren(expressions: IExpression[]): IDictionary<string> {
22 + const nameValueMap: IDictionary<string> = Object.create(null);
23 + let valueCount = 0;
24 +

PR 提交后, 就可以清楚的看出来代码做了哪些改动, 其他人就可以针对每一行代码写评论提出修改意见。如果确认代码没问题了, 就可以通过代码审查。

接下来还剩下自动化测试的问题。这时候该 CI (持续集成) 出场了。

如果你不了解 CI 是什么, 可以把它想象成一个机器人, 每次你提交一个 PR (严格来说是 Commit, 这里略作简化) 到源代码服务器, 这个机器人马上就知道了。

然后它创建一个干净的运行环境, 把你提交的代码下载下来, 再下载安装所有依赖项, 然后运行你的所有测试代码, 运行完后, 把测试结果报告给你。测试结果直观的反馈在 PR 上, 绿色表示通过, 红色表示不通过。

video-react / video-react

Unwatch

36

Unstar

1,134

Fork

183

<> Code

Issues 42

Pull requests 1

Projects 1

Wiki

Insights

Settings

New feature: Customizable delay time for auto hide of ControlBar #233

Open

mondaychen wants to merge 2 commits into master from control_bar_active_time

Conversation 1

Commits 2

Checks 0

Files changed 5

+23 -4

mondaychen commented 10 hours ago

Member

+

⋮

Implements #146

Initially I added `controlBarActiveTime` on `Player` level. However, I think it's much more intuitive to move this prop to `ControlBar` level. I used a trick to read it from children in `Player` so it can be a prop of `ControlBar`.

Let me know if you have any concerns.

mondaychen added some commits 4 days ago

Add ``controlBarActiveTime`` prop

Put `autoHideTime` in `ControlBar` and read it in `Player`

mondaychen requested review from JimLiu and xiaoyuhen 10 hours ago

JimLiu approved these changes a minute ago

View changes

JimLiu left a comment • edited

Member

+

⋮

LGTM

Reviews

JimLiu

xiaoyuhen

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because your review was requested.

2 participants

Lock conversation

Add more commits by pushing to the `control_bar_active_time` branch on `video-react/video-react`.

Changes approved

1 approving review [Learn more](#)

Show all reviewers

All checks have passed

2 successful checks

Hide all checks

continuous-integration/travis-ci/pr — The Travis CI build passed

continuous-integration/travis-ci/push — The Travis CI build passed

This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

CI 运行结果

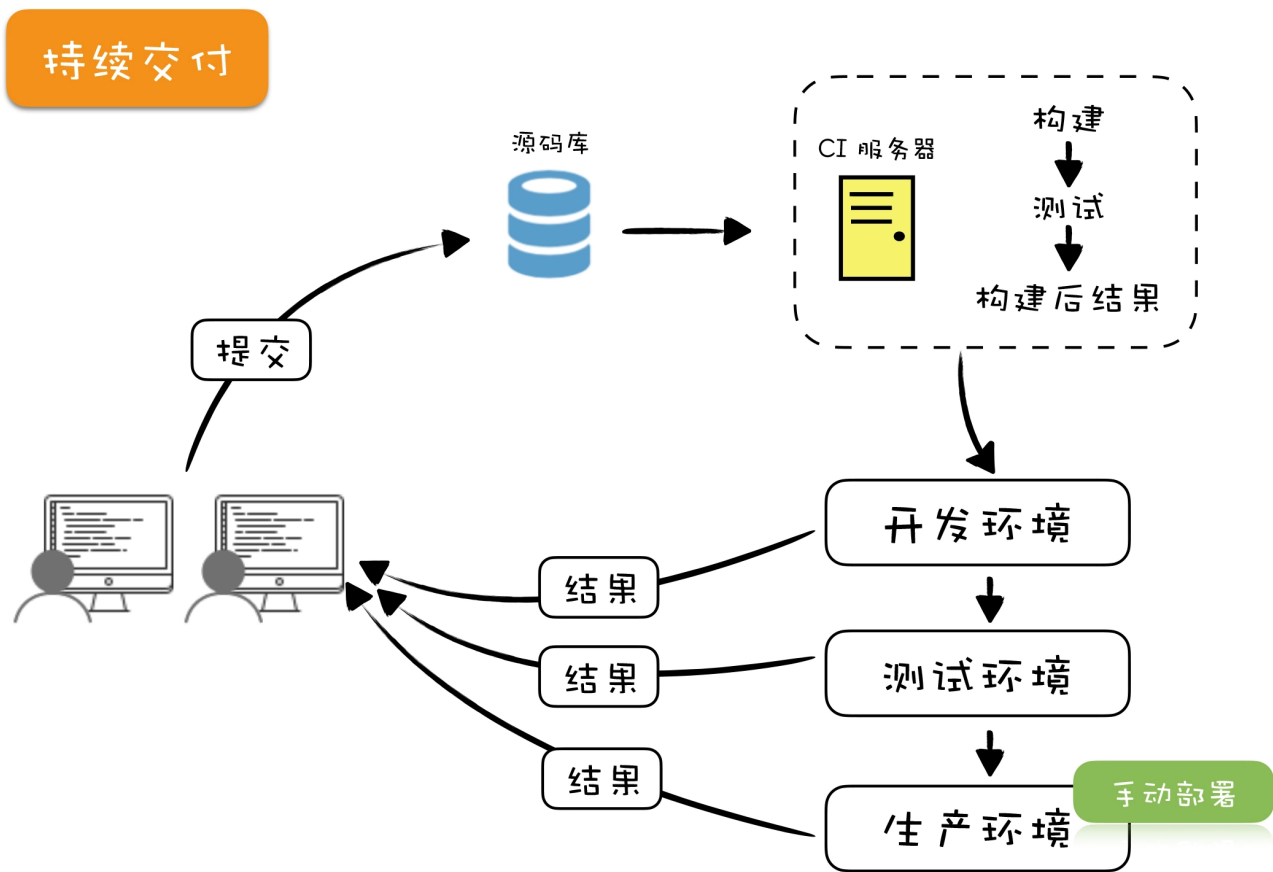
图片来源：Video-React项目PR

关于 Git 和 CI，我在之后的文章中会展开讲解，这里只是为了展现敏捷开发方法的流程。另外，阮一峰老师写过两篇文章，《Git 工作流程》《持续集成是什么？》，你也可以先行

阅读了解。

至此，代码审查和自动测试的问题都解决了。当一个 PR 代码审查通过，以及 CI 通过了所有自动化测试，就可以合并到 master 了，而且我们也可以认为合并到 master 后的代码也是稳定的。

至于自动部署测试环境，反倒是简单，就是 CI 这个机器人，在你代码合并到 master 的时候，再次运行自动化测试代码，测试通过后直接运行自动部署的脚本，把 master 代码部署到开发环境或测试环境上。



在这里以一个开发任务为例，大致讲解一下应用敏捷开发方法的基本开发流程：

把要开发的 Ticket 从 “To Do” 栏移动到 “In Progress” 栏；

从主干（master）创建一个分支（branch），基于分支去开发功能或修复 Bug；

编写实现代码和测试代码（单元测试和集成测试），是不是测试驱动不重要，看个人偏好或团队要求；

持续提交代码更新到分支，直到完成；

创建 PR（Pull Request，合并请求），邀请其他人帮忙 Review 代码，根据 Review 的结果，可能还需要更新几次；

CI 在每一次提交代码到代码库后都会自动运行，运行后主要做这些工作：

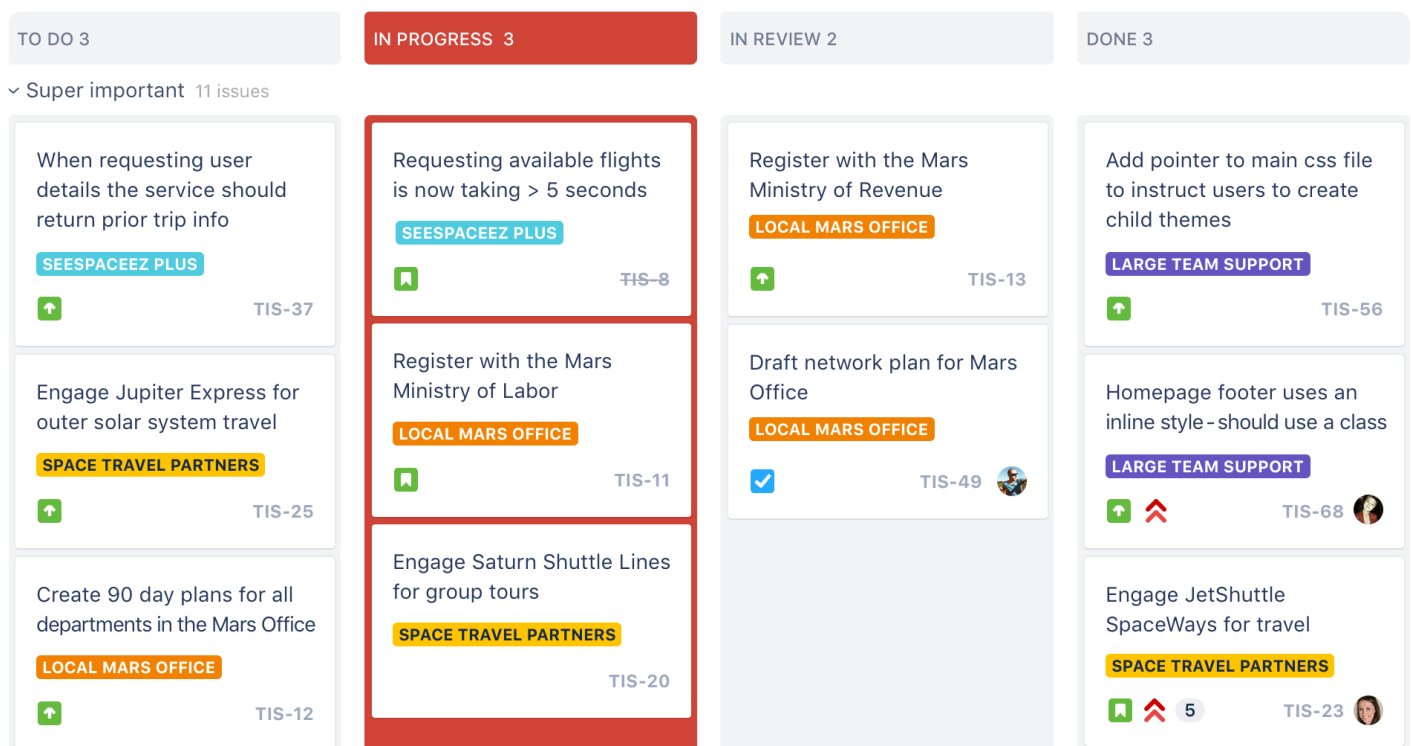
- 检查代码格式是不是符合规范；
- 运行单元测试代码；
- 运行集成测试。

最终这些检查都完成后，CI 会把执行结果显示在 PR 上。通常绿色表示通过，红色表示失败；

PR 能合并需要满足两个条件：CI 变绿 + 代码 Review 通过；

PR 合并后，CI 会自动构建 Docker Image，将 Image 部署到开发环境；

将相应的 Ticket 从看板上的 “In Progress” 栏移动到 “Done” 栏。



图片来源：Jira

正常来讲，你是需要严格遵守开发流程的，但偶尔肯定也有紧急任务，来不及写测试代码，这种情况下，一定要再创建一条 Ticket 跟踪，以确保后续完成测试代码。

部署上线流程

最早的时候，程序员都是自己管服务器，但是由于这样过于随意，就会导致很多问题出现。

于是后来有专门的运维团队，将开发好的程序，编译好，数据生成脚本写好，然后写成部署文档，交给运维去手动部署。这个过程无比繁琐、无比慎重，通常几周才部署一次，遇上打补丁才隔几天部署。

这些年随着容器化、微服务、DevOps 这些技术或概念的兴起，部署已经变得越来越高效，大厂已经开始在部署流程上融合这些理念。

以前是运维人员按照文档部署，现在已经变成了 DevOps 写自动化部署工具，然后开发人员自己去部署生产环境。

现在大厂的部署也都实现了自动化，但是流程上还是有一些控制。

首先，部署的不再是程序代码，而是 Docker 的 Image，每次代码合并后 CI 都会自动生成新的 Image，测试也是基于 Image 测试。

部署生产环境之前，先在内部的测试环境充分测试。

部署生产环境前，需要审批确认，有 Ticket 跟踪。

部署时，先部署一部分，监测正常后再全量部署。

整个过程都有监控报警，出现问题及时回滚。

如果一切顺利的话，整个生产环境的服务部署过程通常几分钟就完成了，这在以前简直是不敢想象的事。

每日站立会议

在敏捷开发中，每日站会是非常有名的。在大厂，但凡实施敏捷开发的小组，上班第一件事，就是一起开一个站会，沟通一下项目的基本情况，这也导致会议室越发紧张起来。

虽然站立会议什么时间开都可以，但是早上无疑是最好的时机，一天工作的开始，开完会全身心去干活。

是不是站着开会其实不重要，重点是要高效沟通反馈。开会时间控制在半小时以内，半小时内不能完成的应该另外组织会议。

谁来主持站立会议呢？在敏捷的 Scrum 中，有一个角色叫 Scrum Master（敏捷教练、敏捷大师），主要任务就是保证各种敏捷流程的。

所以通常是由 Scrum Master 主持会议，也可以采用轮班制，每个星期换一名团队成员主持。负责主持会议的人，主要职责是组织会议，一个一个环节开展，控制好会议节奏。

开会都干什么呢？主要有三个话题：

1. 成员轮流发言

每个人轮流介绍一下，昨天干了什么事情，今天计划做什么事情，工作上有没有障碍无法推进。

一个成员的发言可能是这样的：“昨天我实现了用户登录模块的前端输入框，今天打算完成后端 API 调用，在实现后端的时候需要 API 组的支持，昨天发现他们文档有问题，不知道该找谁。”

要注意的是，这过程中很容易偏离主题，比如突然有人提了一句：“我们好久没团建了，是不是该出去玩玩了。”很可能大家都很 high 的讨论起来了，这时候会议主持者要及时打断，记录到“问题停车场”，让下一个人继续，先保证大家能高效完成这一环节。

问题停车场（Parking lot question），把需要进一步讨论的问题暂时放到这里，一会儿再讨论。

通过这样的形式，项目成员可以相互了解任务进展，有困难也可以互相支援，及时发现问题和风险。还有一个重要因素，就是每个人对于自己提出的目标，也会信守承诺，努力完成。

2. 检查最新的 Ticket

前面提到所有日常工作都是基于 Ticket 来开展的，这些 Ticket 可能是测试报出的 Bug，也可能是产品经理提交的需求，也可能是其他。

所以每天例会都需要检查一下新增的 Ticket，并且要甄别一下优先级，然后决定是放到当前 Sprint，还是放到 Backlog（任务清单）。

这个阶段同样要注意不能发散，不要针对 Ticket 的细节展开过多讨论，有需要讨论的同样可以先收集到“问题停车场”，会议组织者需要做好控制。

3. 停车场问题

在这个环节，大家可以针对之前来不及讨论的问题进行讨论，能在会议时间内解决的问题，就马上解决，不能解决的会后再私下讨论或者再组织会议。

当然，大厂的流程规范还有很多，在这里我仅列出与敏捷相关的主要开发流程。

总结

我们知道，在敏捷开发中有很多概念，像 Backlog、持续交付、每日站会等，这些概念最终要变成实践的话，就必须要通过一定的流程规范来保障这些概念的实施。

这就是为什么很多公司写代码要求你写自动化测试代码，为什么要用一些像 Jira、禅道这样的项目管理软件来管理任务，为什么要每天开站立会议，为什么要有代码审查。这些都不过是为了保障敏捷的实施。

如果你在实施敏捷开发的项目工作，就可以多去观察平时工作中这些和敏捷有关的流程规范，再结合敏捷开发中的知识点，就能很好的帮助你理解敏捷开发，理解这些流程规范背后的理论依据。

如果你工作中不是用的敏捷开发，也可以参考本文中提到的一些实践，尝试着试用起来。

在下一篇里，我还会以一个具体的项目小组对敏捷的应用为例，继续给你讲讲大厂都在用的那些敏捷方法。

课后思考

你的项目中，有哪些跟敏捷开发相关的实践？你觉得哪些做的好的地方，哪些做的不够好的？或者哪些是你疑惑的地方，都可以留言讨论。另外，你可以再思考一个问题：一个每周一个 Sprint 的敏捷项目，怎么保证每周都有交付，还能保证产品质量？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (35)



纯洁的憎恶

2019-03-07

分治策略是应对庞大复杂系统的惯用思路，但它的难点或精髓在于如何确保形散神聚。

详细计划（甘特图）VS任务状态（Ticket）

代码不稳定&环境部署麻烦

VS

代码审查&自动测试&自动部署（GIT、CI、DevOps）

上传下达VS频繁沟通、提醒、分享

大厂的敏捷开发实践，把枯燥的编码变得跟玩游戏一样。借助有效的流程与工具，能够有效节约团队成员的精力，聚焦于任务或角色，不会因频繁“统一思想”导致“技术动作变形”。而另一面，在大厂里每个人通常都是螺丝钉，长此以往也许会养成不谋全局的习惯。如果能从自己的角色中跳出来，俯瞰整个组织协作的全过程，并站在这个视角上思考问题，一定会有更喜人的收获。

作者回复: 感谢你帮我把螺丝钉这段写出来了, 我有想写这么一段类似的, 后来觉得篇幅太长另外和软工关系不那么大就没扯淡了 😊

大厂优势就是资源多, 项目大, 如果如你所说, 能跳出所在岗位, 站在全局多观察多思考, 能学到很多东西 👍

如果只局限于自己岗位做一个螺丝钉, 只会大厂的一套很窄领域的技能, 时间一长很可怕, 将来出去后难以找到合适的工作!



👍 37



J.M.Liu

2019-03-07

老师, 敏捷开发这么强调扁平化, 这么重视人, 这么强调开放而弱化约束, 那和最初没有软件工程时期的开发主要区别是啥呀 😊

作者回复: 好问题, 你难倒我了 😊

前面介绍过, 没有软件工程的时候呢, 开发就是边写边改模式, 没有需求分析, 没有架构设计, 没有测试, 导致很多问题。

敏捷开发的话, 需求分析还有, 只是简化了, 架构设计也有, 只是说仅仅做当前Sprint的架构, 测试也没省, 只是很多自动化测试辅助, 所以让测试人员从繁重的体力劳动中解放出来很多。

共 3 条评论 >

👍 18



Felix

2019-03-07

“一切以Jia为准”一直我长挂在嘴边的一句话, 以此也教育了用户(开发、产品、测试)养成习惯, 之后大家也乐于通过Jira来沟通、对齐信息

作者回复: 👍是的, 一切以Ticket为准! 不用担心问题被遗忘, 会被跟踪, 直到被解决或决定不解决!



👍 12



Charles

2019-03-07

创业公司，目前只做到了

1. 所有需求、bug、任务等都放issue里
2. 项目经理发起，大家讨论，结合市场或阶段性目标整理这个版本包含哪些issue，评估时间，再进入开发阶段
2. git管理代码，有master、develop以及bug或特性分支
3. master对应生产环境、develop对应测试环境，修复bug或特性，本地自测完配合一点点的单元测试，推送到develop自动部署到测试服务器，测试开始测试，测试通过再把对应的bug或特性分支合并到master自动部署生产环境

看了老师的专栏，感觉可以超几个方向努力更敏捷

1. 提高单元测试覆盖率，尝试自动化测试，目前人工测试效率低，压力大
2. 测试环境和生产环境容器化，目前各种特性并行开发测试环境容易冲突
3. master分支尝试pr的方式，互相阅读代码学习还能发现一些单个人考虑不全测试又测不到的潜在隐患

就是这些做完不知道是否又会引入新问题，感谢老师专栏，学到很多

作者回复：你说的这几个方向都很好的。第二个要慎重一点，如果没有特别懂的人不要着急马上去做，会有不少坑要填。一三可以尝试做起来。

自动化测试的话，最重要一点，就是要让开发同时写一些自动化测试，传统测试人员短期内恐怕难一下子写出来高质量自动化测试代码。还有就是要项目管理上要支持，例如说写测试的时间要放到计划里面，必须要留出来写测试的时间，不能不给又想马儿跑得快不让马儿吃草。

PR是个很好的代码审查以及结合CI看测试状态的方式。但是需要花点时间去搭平台。如果创业公司，没有必要自己去基于Jenkins这种搭建，去买成熟的第三方服务就足够了。

最后一点，要改变，不要着急一蹴而就，一点点去试点，然后再推广，不建议贸然改变太多。

共 2 条评论 >

👍 9



KK_TTN

2019-03-07

如何培养自己维护ticket的习惯呢？感觉写代码是一件愉快的事情，倒是经常会忘记(或者内心去回避)更新ticket的状态

作者回复: 这倒是小事了, 可以养成习惯, 每天下班前检查一下, 一次性更新。

我有写过一个小脚本, 在CI里面运行, PR merge或者部署测试环境都自动更新, PR里面标题加上Ticket编号就可以。又酷又省心👍



👍 7



敏捷教练夏勇杰

2019-08-21

之前也在团队里实施过Code Review的机制, 但是, 大家对于Review别人的代码都不是很积极, 最后就变成了Team Leader一个Review所有人的代码, Team Leader没有时间做这个事情的时候, 大家就敷衍了事, 直接通过, 让Code Review流于形式。对于这种情况, 宝玉老师遇到过么, 是如何解决的?

作者回复: 我有写过一篇《Code Review最佳实践》发在了博客园:

<https://www.cnblogs.com/dotey/p/11216430.html>

供参考



👍 4



卡皮

2019-03-14

敏捷开发中有什么好用的工具推荐呢?

作者回复: 具体看哪一类的工具, 比如CI的话你可以看看开源的Jenkins (<https://zhuanlan.zhihu.com/p/54050436>), 比如项目管理工具你可以看看Jira或者类似的, 自动化测试具体要看你的语言, 源代码管理GitHub



👍 4



javaadu

2019-03-07

思考题: 一周一个sprint, 要保证每周交付的话, 一要看ticket的粒度(任务拆分)是否合理, 二要提前一周做计划, 三要每周结束做总结。

作者回复: 任务拆分力度可以解决好分工协作问题; 敏捷里面每个Sprint开会前会有迭代计划会, 团队成员一起安排计划下一个Sprint的ticket; Sprint结束还有迭代回顾会做总结。

交付的问题解决了, 质量上再思考思考?



alva_xu

2019-03-07

在一个以Scrum 为方法的敏捷团队里,
首先, Scrum master是呵护develop team的保护神, 他的其中一个职责是保护每一次迭代的工作量是dev team能按时完成的, 而且保护dev team 能专注于现有sprint back log的实现, 不会被其他干系人的新需求所打断。
其次, Dev team是一个T型团队, 技术比较全面, 许多事情多能自助搞定, 比如, 开发人员同时又有测试技能, 同时如果结合结对开发, 测试驱动开发, 那么, 交付物的质量就更有保障。再者, 在一个敏捷团队里, 人数比较少, dev team的沟通能力都比较强, 沟通可以比较充分, 所以解决问题的能力就比较强, 工作效率比较高
最后, 敏捷模式的开展, 也依赖于工具的使用, 目前常用的CICD工具, 与jira/confluence 需求沟通管理工具的打通, 部署次数的提高, 无疑大大提高了开发发布效率, 同时也提高了发布质量。
综上所述, 只要在人员组织架构、工具产品、流程这三个方面都达到了敏捷的要求, 那么发布质量就有了保证。

作者回复: 对敏捷了解已经很熟悉很深入了👍

你还可以继续考虑考虑还有没有其他手段可以加强质量的? 尤其是可以从瀑布模型那边学习借鉴一些。



小老鼠

2019-08-21

- 1、小厂如何使用敏捷? 好些小厂朋友抱怨敏捷玩不起来或不好用。
- 2、像嵌入式软件等非BS架构产品可使用容器+微服务吗?
- 3、以前我测试过一款网络管理产品, 走的是SNMP协议, 但由于各个客户所用产品的厂商不同, 比如A用户用华三交换机、B用户用华为交换机、C用户用华中兴交换机、D用户用阿朗交

换机…，各厂商交换机除了支持标准SNMP协议外，还支持自定义协议，所以该产品测试非常难。现在在DevOps 下可以解决吗？

作者回复：小厂使用敏捷开发，几件事需要做起来：

1. 固定迭代周期，每2-4周一个迭代，迭代结束能交付可用的产品，为了定期发布舍得砍掉功能需求
2. 增加自动化测试的比例，把源代码管理、CI（持续集成）用起来，借助开发流程、自动化测试保证基本产品质量
3. 用好任务管理系统，把所有要做的任务都跟踪起来，按照优先级都排到相应的迭代版本，在一个迭代中，有紧急任务插队加塞，相应的就应该把其他任务移出去。

嵌入式软件能不能用微服务这问题超纲了，我也不懂

DevOps也不是银弹，主要有三点你可以借鉴：

1. 高度自动化
2. 透明可量化
3. 紧密协作

你这个问题可以从上面几个角度考虑，尤其是自动化的角度考虑是不是可行。



👍 3



舒偌一

2019-03-09

好的地方是项目透明，对项目情况比较了解，如果成员责任心好，那就很赞。缺陷是外在事务的干扰，我们现在的做法是，有一个人在一个sprint内不参与，专门处理意外情况。希望老师给一个建议。

作者回复：你说的应该是Scrum Master的角色，你们这个是很好的方案，必然要有人去处理对外的对内的杂事，保证其他人可以专注的工作。

另外在后面一篇也介绍了一个轮值的方案，就是每周安排一个人专门去做一些杂事，大家轮流来。



👍 3



Tiger

2019-03-08

在敏捷里面，开发写自动化脚本测试，那是不是就不需要测试这个角色了啊？感觉在敏捷里面，只需要开发这一个角色就可以了啊？

作者回复: 在下一篇里面还会有谈到这个问题。自动化测试是辅助的, 还是离不开人工的测试。而且开发写的集成测试和测试写的自动化测试还是有一点差别, 一个是用程序模拟的操作的模拟的固定数据, 而测试则用的是真实的数据真实的环境。

举个例子来说, 网页的自动化测试, 开发只会用Chrome Headless, 数据都是事先写好的模拟数据; 测试的话会用主流的Chrome、Safari、Firefox、Edge分别测试(自动化或手动), 数据都是测试环境的真实数据。



Xunqf

2019-03-08

我们是小厂, 但是也是在尝试敏捷开发, 老师提到的我们基本上也都做了, 说一下做的不足的地方吧, 开会解决问题容易搞成头脑风暴, 然后开不出结果, 然后因为是敏捷开发, 老板和产品经理总是随意的对需求增删改查😂😂😂

作者回复: 可以试试“问题停车场”, 把问题留在后面在讨论, 并且严格限制会议时间。

对于需求更改, 严格做到当前Sprint不做更改, 另外通过迭代计划会, 一起确定下个Sprint的计划, 要做的事情。



Felix

2019-03-07

Git方面也要求团队Master中的代码必须通过Merge Request(Pull Request)来, 也作为Code Review的最后一道关卡

持续集成方面大部分通过Jenkins、几个微服务是通过Gitlab CI, 我们的终极目标是基于镜像部署发布, 屏蔽环境影响

作者回复: 是的, 基于PR结合CI和Code Review是一个非常好的控制签入代码质量的手段!



技术修行者

2019-12-07

在某大厂工作很多年了, 一直在用Scrum来管理团队, 遇到的做的不好的地方挺多的, 边实

践边改善吧。

1. Scrum Master定位不清晰，Scrum Master应该是帮助开发团队屏蔽各种干扰，保证团队高效能的角色，但有时他会和PO站在一起，不断调整Sprint的scope。
2. 站会流于形式，开发团队为了站会而站会，不能很好的在团队内部保持信息透明。
3. 团队在多个地方，有时不在一个时区，沟通成本和效率都不好。

如果想要每周一个Sprint，最关键的是团队文化的建设，让整个团队认可敏捷开发，另外对scope做严格管理，需要提前做计划，不能等Sprint开始之后再去讨论Scope。

作者回复: 👍很好的分析！

有些问题是可以改进的，比如1和2，但3确实有难度，首先要凑一个一起沟通的时间都难，另外通过会议系统沟通确实不如面对面沟通效率高。

所以通过微服务这种架构将大组拆分成小组，分组时，尽可能一个城市的在一个组，这样可以有效改善沟通问题。



👍 1



hua168

2019-03-11

一直跟着老师的专栏走，看着留言很激动~~我想问一下：

1. 像我运维刚刚学开发没项目经验怎办？没有地方可以上手实验呀？
2. 敏捷开的有没有好用的免费开始的管理工具，jira是收费的？

作者回复: 1. 不知道你是不是已经工作并参与到实际项目中

建议先多观察身边的项目，对你学到的知识加以印证，项目经验其实更多来源于参与后的观察和思考。

举例来说，你可以结合你学到的知识，看看你们项目现在是什么开发模型，看看项目经理是如何处理项目中的问题的。

如果还没参与到实际项目，可以做点开源项目练练手。

2. 除了Jira有很多选择，例如禅道、Worktile，你搜索一下：“项目管理软件”，腾讯、阿里、华为都有自己的



👍 1



2019-03-10

宝玉哥 欢迎来到amazon，工程师负责设计开发测试DevOps一条龙，每一个人都欢迎对整个环节的每一个部分提建议。

PS: engineer的responsibility也和他的level有关。

作者回复: 哈哈，你们大亚麻是大厂中的超级厂👍

厉害的，一栈到底！



👍 1



舒偌一

2019-03-09

review很重要，组内有一个持续改进的review清单，依据清单做。

作者回复: Review不仅仅是可以发现很多问题，另外如果知道自己的代码会被Review，也会写的认真一点：)



👍 1



一路向北

2019-03-08

保证每周都有交付，首先团队成员必须得充分认同敏捷开发的理念，并且有相关的知识培训，其次是项目负责人对每一个需要交付的Sprint的分解到位，团队成员之间相互沟通的路畅通，再一个是对每次的站立会议落到实处。

这节课的内容丰富，需要一段时间的消化，我也试着将敏捷的模式应用到实际的开发项目中。

作者回复: 赞👍

可以在一个点一个项目先应用起来，慢慢再更多内容。另外限于篇幅，很多内容并没有介绍特别详细，可以辅助阅读一些书籍和文章，可以帮助你更好了解。



👍 1



dancer

2019-03-08

问题停车场真的很有必要，好多时候沟通进度阻碍的站会变成了问题讨论会。

作者回复: 是的，开会要高效，就不能太发散，问题停车场是一个很好的方式改进这个问题。



1