我是如何开始“极限编程（XP）”的？

作者：不详
翻译：东东[AKA]

显而易见“极限编程”必须从一个新项目的开始就应用。开始的时候要收集一些用户的信息和那些操作那些看起来比较危险的问题的解决方法。仅仅花几个星期来做这件事。然后安排一个计划会，在这个会议期间可以组织一个计划方案的竞赛。邀请客户、开发者和管理者制定一个大家都同意的计划表。然后启动你的迭代开发，迭代计划会也要同时进行。现在你已经开始了。

通常只有当一个项目陷入困境的时候才会去寻求像 XP 这样的新的软件设计方法。在这种情况下，使用 XP 的最好的方法是认真的思考一下你当前的软件方法，并断定出使项目慢下来的原因（如果是这样，在应用 XP 之前你最好认真的回顾一下你现在应用的软件方法，并且判断出是整个项目放慢的原因）。然后首先对这个问题使用 XP 方法。

例如：如果你发现针对你的需求规格进行的开发中有 25%是完全没有用的，那么停下来，那么你应该坐下来和你的客户进一步讨论，并写下用户的意见。

如果你的问题在于需求的不断变化导致你不得不经常重建你的计划，那么尝试一下计划编制竞赛（但是你首先需要用户信息(user stories)）。尝试一种开发的迭代风格并且及时地制定设计任务的风格。

如果你的最大的问题是产品中的大量 bug，那么尝试一下自动功能测试。用这种测试套件用于回归和有效性测试。

如果你的最大的问题是集成错误，那么尝试一下自动的单元测试。在任何新代码发放到代码仓库之前，要求所有的单元测试要百分之百的通过。

如果一两个开发者成为瓶颈（因为他们拥有系统中的核心类并且必须进行完全的修改），那么尝试一下集体代码所有权(collective code ownership)（你也需要进行单元测试）。只要他们需要，可以让每个人都对核心类进行修改。

你可以不断地尝试这种方法，直到解决所有问题。然后尽可能的不断地尝试剩下的练习。你加的第一个练习可能很简单。你正在解决一个大问题，却只付出了一点点额外的努力。第二个练习可能也很容易。但是在应用 XP 的一些规则和所有规则之间有一个点，你可能必须坚持才能做到。你的问题可能已经解决，并且你的项目可能还可以控制。坚持旧的熟悉的方便的开发方法看起来可能比较好，但是如果你继续应用 XP 的话，在项目要结束的时候你肯定会得到回报。你的开发团队会比你想象的更加有效率。当你达到这一点的时候，你就会发现 XP 的规则已经不仅仅是停留在纸上了。这些规则之间还有一些协作关系，如果你不深入进去是很难理解这些关系的。

这种如同爬山的方式（up hill climb）对于同时开发两套系统（pair）是特别有效的，但是这种技术的回报是很大的。单元测试也会花费一些时间，但是单元测试是在其他 XP 实践的基础上，所以回报仍然很大。

采用 XP 技术的项目是不安静的；看起来在项目中会有人在一起讨论问题。人们四处走动，彼此提出问题，相互讨论设计方案。人们为了解决一些难题而自然的碰面，然后又自然的分开。应该鼓励这种交

流的过程，提供一个会面讨论的地点，并且设置这样的两个人能够轻松的一起工作的环境。这种完全的开放的工作环境能够鼓励小组间的交流。

自由、协作、创造 — 为了明天
“来自大雪山的大雁阿卡”

更多精彩文章，请访问：http://www.AKA.org.cn **精彩文章 栏目**
本文如有翻译错误或不妥，请 Email **至** AKAMagazine@yahoo.com

附件：英文原文

How do I start this XP thing?

The most obvious way to start extreme programming (XP) is with a new project. Start out collecting user stories and conducting spike solutions for things that seem risky. Spend only a few weeks doing this. Then schedule a planning meeting during which the planning game will be used. Invite customers, developers, and managers to create a schedule that everyone agrees on. Begin your iterative development with an iteration planning meeting. Now you're started.

Usually projects come looking for a new methodology like XP only after the project is in trouble. In this case the best way to start XP is to take a good long look at your current software methodology and figure out what is slowing you down. Add XP to this problem first.

For example, if you find that 25% of the way through your development process your requirements specification becomes completely useless, then get together with your customers and write user stories instead.

If you are having a chronic problem with changing requirements causing you to frequently recreate your schedule, then try the planning game. (You will need user stories first though.) Try an iterative style of development and the just in time style of planning of programming tasks.

If your biggest problem is the number of bugs in production, then try automated functional tests. Use this test suite for regression and validation testing.

If your biggest problem is integration bugs then try automated unit tests. Require all unit tests to pass (100%) before any new code is released into the code repository.

If one or two developers have become bottlenecks because they own the core classes in the system and must make all the changes, then try collective code ownership. (You will also need unit tests.) Let everyone make changes to the core classes whenever they need to.

You could continue this way until no problems are left. Then just add the remaining practices as you can. The first practice you add will seem easy. You are solving a large problem with a little extra effort. The second might seem easy too. But at some point between having a few XP rules and all of the XP rules it will take some persistence to make it work. Your problems will have been solved and your project is under control. It might seem good to abandon the new methodology and go back to what is familiar and comfortable, but continuing does pay off in the end. Your development team will become much

more efficient than you thought possible. At some point you will find that the XP rules no longer seem like rules at all. There is a synergy between the rules that is hard to understand until you have been fully immersed.

This up hill climb is especially true with pair programming, but the pay off of this technique is very large. Also, unit tests will take time to collect, but unit tests are the foundation for many of the other XP practices so the pay off is very great.

XP projects are not quiet; there always seems to be someone talking about problems and solutions. People move about, asking each other questions and trading partners for programming. People spontaneously meet to solve tough problems, then disperse again. Encourage this interaction, provide a meeting area and set up workspaces such that two people can easily work together. The entire work area can be open space to encourage team communication.