# 转 一个高效成熟的软件开发流程和团队

下面是开发管理最规范一家软件公司的开发流程，该公司总部位于美国硅谷，其开发的产品曾获得 PC Magazine 的最高五星级的优秀好评。供大家参考。

## 1. 项目计划

在一个产品发布并使用之后，其中肯定有许多地方不如意和值得改进的地方。客户在使用的过程中会发现一些问题，提出更高的需求，市场也在发生变化，我们的竞争对手也在发展，新的技术不断地产生，这些因素推动着我们的产品不断地向前发展，使它的版本不停地往上增长。这些发展的需求不是一下子提出来的，在客户使用的过程中发现某些不如意不方便的地方，他们会向我们的技术支持人员提意见，而技术支持人员会把这些需求以 BUG 的形式存入 BUG 数据库中，其级别一般定义为下一个版本的 Feature。有些上一个版本未解决的 BUG 也可能需要在本版本中来解决。因此当我们来开发下一个版本时，其许多特性已经存在于 BUG 数据库中了。当然新版本的特性不是只从 BUG 中获得，管理层可能从市场的角度来提出新的特性以求领先竞争对手，开发人员本身也可提出某些要求来纳入新版本开发的计划中，如要求对某部分代码进行重构以使其结构更清晰更容易维护，执行效率更高。

每个人把同自己相关的功能模块收集起来，同时预估时间，其中主要包括写文档的时间、开发时间和单元测试的时间，一般要求精确到工作日。这些信息发送给组长，组长再把本小组人员的任务和预估时间发送给管理层，由管理层对此任务及进度进行评估审核，管理层会根据产品发布时间及客户需求、市场因素等方面作出选择，可能某些功能由于时间紧急会被推迟到下一个版本中去。若预估出来的时间同预计的产品发布时间有较大冲突，而且此功能是本版本中必须得做的，则开发小组会被要求重新预估时间，加快开发速度来达到这个要求。

虽然这个开发进度时间是一个大概的估计时间，但我们要尽力按照这个开发进度来执行。每个星期五下午我们有一个 Status Meeting（一般那时工作效率较低，适合开会），在此会议上我们会根据这个进度来 review 我们的工作，每个人手上的工作是否按照这个进度在走，是否有人延后了，是否 block 住别人的工作了。在此会议上每个人都要报告自己的进度，同时还要报告上个星期做了什么，正在做什么，以及下个星期打算做什么。通过这个会议，会让你觉得有人在监督你，无形之中迫使你不断地督促自己不要使任务延后，如果有延后的迹象也会尽早发现而赶上。若某些经过努力不能赶上，那也没有办法，只能修改原先的进度

表，因为那是我们的估计与现实发生了偏差，我们必须使我们的进度表符合实际情况，这可以避免许多项目发生最后的 20%的工作量会占据 80%甚至一直拖后的情况。修改进度表的情况我们曾经发生过，有一次在按照原先的进度执行到将要完成的状态时突然接到通知由于市场及客户的原因要求加入另一项重大的功能，这个功能对我们程序的结构有非常大的影响，因此我们就要重新制定一个进度来满足需求。在这种情况下，产品原先的开发进度被打乱，发布时间也因此推迟。当然这种情况应当尽力避免，尤其在项目后期产生新的需求，若不得已也应重新规划进度，而不是仍旧依照原先的进度去执行，因为老的进度已不能反映现实的情况。

### 2. 开发文档

在项目进度安排中我们已经把写文档的时间也规划进去了，这里虽然是写文档，其实是设计程序，整理一下思路与架构，磨刀不误砍柴工，这样在实际写代码时会流畅很多，节省时间，因此可以说真正有思想性的东西都在写文档这段时间内完成了。当然我们这里的文档格式不象 ISO 那样规定了条条框框，我们的文档格式相对自由，基本上能随意发挥，但对于几个主要点一般来说是需要说明的。要求写的文档能让他人比较容易地看明白，能把问题讲清楚，能反映你的设计思想。文档的数量也不多，开发文档有两类，一类是 function Spec，另一类是 Design Document。

function Spec 中需要写明的是本模块完成的任务，解决什么问题，有什么作用，为什么要这些功能，此外我们还会添加进适用范围，有什么不足，注意点是什么，还有哪些地方在以后可以进行改进。在这个 function Spec 中不涉及到任何非常详细的算法。此文档不光给开发人员看，还让 QA 及其他成员以及后来的新人能根据此文档来了解此模块的大致功能，同时也会给文档编写者看，他们会根据这些 function Spec 整理出一份用户手册，告诉用户此版本中新增了哪些功能，各功能模块有什么作用，如何使用等信息。因此在我们的开发过程中 function Spec 是很重要的文档，此文档完成后会抽出一段时间同相关人员及 QA 一起 review 这个文档，让 QA 了解设计者的意图，同时熟悉新的功能模块，为接下来的测试作准备。如果其中有误解或不明之处，大家会提出来探讨并由开发者修正。

Design Document 中主要描述实现此模块所涉及到的主要算法、数据结构、类的层次结构及调用关系。这个文档的阅读者主要是开发人员，包括任何想了解详细实现代码的人，帮助人们理解代码。在某些功能模块比较简单的程序中，此文档所描述的信息会比较少。此文档不象 function Spec 要在开始写代码前就编写完成，它可以随着代码编写的进行而增加，但基本上遵循文档先行原则，也就是要增加新的代码或修改代

码前若有涉及到文档部分的应先修改文档，然后再修改代码。

### 3. 编写代码

由于我们用 JAVA 语言进行开发，因此我们借助了 Jbuilder IDE 工具。关于代码风格，我们基本上套用 Jbuilder 中自动的代码格式编排，但其中需要改变的是缩进是 4 个字符，类与类之间间隔 2 行，方法与方法之间间隔 2 行，import 类时用完整的类名。写代码时要对类及函数提供详细的注释及说明，基本做到看它们的说明就能知道这个类或函数的功能以及主要算法的实现原理。在开发过程中对主要的模块要编写 UnitTest，同时要 UnitTest 先行，也就是遵循 XP 规则中的测试驱动原则，当所有的单元测试代码通过时，此功能也就基本上完成了。

### 4. 代码管理

我们采用 VSS+SourceOffsite 进行版本控制，其中存放了此产品的所有源代码、库文件、文档及 release 时的安装程序，各个部分存放在不同的目录中。每天早上要求开发人员从 VSS 中 get latest version 的源代码，然后进行编译并开始一天的工作。在下班之前理论上要求员工 check in 所有当天修改的代码，在 check in 之前要保证编译是能通过的。若有谁 check in 的代码导致 daily build 失败则会被要求某些惩罚措施或警告，象微软公司要负责照看当日的每日构建。有时我们编写的代码涉及到多个文件，而且此改动是比较复杂需要花费多天的工作量，如果现在 check in 进去可能会导致 BVT（Build Verify Test）测试通不过，因为有些代码没有完全完成，而之前的代码能使 BVT 测试通过，而且这些代码基本上不会涉及到他人，在这种情况下可以不 check in 进去，直到全部代码完成能提交 BVT 测试时再一起 check in 进去。

每天我们都会做 daily build，一般是在凌晨 4 点进行，那时有个程序会自动从 VSS 中拉下最新的代码并进行编译。因为我们同美国进行同步开发，因此如果想要把修改的代码进入到这个 build 中去那就需要在凌晨 4 点之前把相应的代码 check in 进去。若有人 check in 进去的代码导致编译通不过则会在本步骤中被发现。当编译完成之后自动产生安装包，测试部门将会对这些代码进行 BVT 测试，同时对 VSS 中开发库打上 label，如果发现了什么 BUG 就能根据这个 label 知道是哪个时候开始出现这个 BUG 的。BVT 是指 Build Verify Test，是对组件中基本功能的测试。这个测试每天都会进行，看新加入的代码或修改是否会影响系统的基本功能，便于及早发现错误。

## 5. 测试

在开发人员完成了 function Spec 后，测试部门开始了测试规划，确定需要测试哪些方面，如何测试及进度安排。测试人员需要写许多测试代码，有些测试代码需要集成进 BVT 测试，有些可能需要进行单独的测试，目的都是为了使产品符合要求，使开发人员容易找出问题所在并改正。产品功能是否符合了要求，是否能被发布是由测试人员决定的，因此测试人员也比较辛苦，责任重大。通过了每天的 BVT 测试，还有一些性能测试、兼容性测试、灾难测试等需要在产品发布前进行。在完成这些测试之后由测试人员决定本产品是否能 release 出去了，如果没有什么问题则会给某些关系较好的用户进行 β 测试，之后再最终 release 出去。

## 6. BUG 管理

由于我们每天进行着测试，因此经常有 BUG 被测试部门发现，一旦发现了新的 BUG，就会被添加进 BUG Tracking System 中。目前较流行的 BUG Tracking System 有 TestTrack、ClearQuest、Bugzilla 等。BUG tracking system 是开发人员和 QA 之间的纽带，开发人员和 QA 通过 BUG tracking system 联系着。每个 BUG 有其类型和级别，预定的类型有
Crash-Data Loss, Crash-No Data Loss, Incorrect functionality, Cosmetic, Feature request 等, 级别有 P1、P2 一直到 P6，它们分别代表了重要性及紧急程度，P1 的 BUG 需要很快 fix，P5 之前的 BUG 在本版本 release 之前必须 fix 掉，若真的不能或不重要则由 QA 确定并降低优先级进入到下一个版本中去 fix。QA 发现一个 BUG 后在 BUG Track 中增加一个 BUG，同时填入相关信息并 assign 给相应的开发人员，开发人员收到 BUG 分析并 fix 后 assign 给 QA 去 verify，其中要填上分析的结果以及如何解决的详细说明。若 QA 对此 BUG verify 通过则 close BUG，否则 verify failed 并重新 assign 给开发人员并等待其 fix。每星期在 Status Meeting 上会进行 BUG 状况报告，主要由 QA 组长报告 BUG 的状况，主要是新增 BUG 数，fix 掉多少，还有多少处于 open 状态，有多少处于等待 verify 的状态，据此可以了解开发及测试情况。有时在 Status Meeting 上我们也会进行 BUG Review，BUG Review 有时是单独一个小组内进行，其主要作用是重新明确每个人头上的 BUG 以及了解每个 BUG 的状况，如开发人员对此 BUG 将作何处理等，以此来了解开发中是否有碰到比较棘手的问题，增加了产品发布风险。在 QA 增加 BUG 和开发人员 fix BUG 的游戏中，BUG 的数量曲线图会象股市曲线一样上下波动，但总体趋势一般是前期 BUG 放量攀升，后期震荡下挫，

若到了后期新 open 的 BUG 数量一直上升则说明风险在增大，有可能无法控制，也就是说 fix 了一个 BUG 导致了多个新的 BUG 产生。在量化开发进度中也可以用代码数量的曲线图来粗略的呈现。在有大量新功能增加时可能代码量的增加会较快，当在 fix bug 阶段，代码的修改较多，因此代码数量的增幅会降低，依据代码量可以看出开发的状况处于何种阶段。

需要指出的是我们对 BUG 的定义比较广泛，一些新功能也可以作为 BUG 被提出，只不过这些 BUG 级别比较低，让它们进入到下一个版本中去实现。因此 BUG 的创建者也可以是技术支持人员、市场人员甚至开发人员本身。关于开发人员本身，因为他可能会找出一些 BUG，有些是其他开发者的，有些可能是此开发者本身的，把这个 BUG 添加进 BUG 库中可以帮助开发人员在以后产生新问题时或类似的 BUG 时有一个借鉴和思路，但此 BUG 的 verify 必须要让测试本模块的测试人员来 verify。

### 7. Code

Freeze 当 P5 之前的 BUG 都被修复了，这时离产品发布日期也就不远了，一般是 2 个星期后就能 release 产品，这时要对 VSS 中的代码进行 freeze，以保证代码库的稳定性。Code freeze 阶段一般会把各开发人员的 check in 和 check out 的权限关闭，若在这时仍有 BUG 报告上来并经讨论确定是重大的且必须在本版本中 fix 的，则需要经管理层同意并特殊地授予权限，在修改完成后修改者要把修改了哪些文件，影响了哪些文档等信息上报给各部门如 QA、build 人员、文档编写者等。在 code freeze 阶段，测试部门在紧张地进行着各种测试，得出各种数据，并决定本版本是否可以 release 了。

### 8. Tech Talk

计算机知识更新速度非常快，经常有一些新的术语、新的名词、新的思想、新的技术所产生，如过离开此行业几个月后重新回来就会对这些新的事物不解，而我们平时为了自己的项目埋头苦干可能忘了周围的世界发生了什么。Tech Talk 就提供了一个让我们了解新知识和最新发展趋势的机会，让大家把知识共享，共同提高。Tech Talk 一般会在项目不是太忙碌的时候进行，主持人会提前一个星期指定某个人去准备一下 Tech Talk，一般此人可能对某方面比较感兴趣，然后他会花一些时间去了解这方面的情况，写成一个文档如 PowerPoint 并上传到局域网内，同时通知大家可以先去浏览。Tech Talk 的内容非常广泛，不一定同我们的项目紧密相关，任何新的思想、新的知识（当然一般是限在计算机领域内）都可作为 Tech Talk 的内

容，而在主讲人讲完之后还有一段时间被大家提问，共同对这个话题进行讨论，答疑解惑。当然 Tech Talk 也可同我们的项目相关，如研究一下竞争对手的产品技术，本公司产品的架构等。研究本公司的产品架构可以使大家对本公司的产品有一个全局的概念，从整体上来看自己的产品，顺便整理一下产品的架构使之更加清晰有条理。平时大家都只注重于自己负责的其中的一小块，在 Tech Talk 中可以跳出自己的小框框来了解全局，同时这也是新员工了解公司核心技术整体框架的好机会。每个模块的负责人需要阐述此模块的方方面面，让大家来了解并回答问题。

### 9. Code Review

当进行工作移交时我们会进行 Code Review，在碰到棘手的 BUG 时也会进行 Code Review，Code Review 是大家了解其详细实现的一个好机会。在 Code Review 之后会对此代码产生亲切感而不是陌生惧怕感，相信很多人在读他人代码时会有非常痛苦的经历，Code Review 是减少此痛苦感的好药方。在进行 Code Review 前，主讲人会提前发出一个通知告诉相关人员要 review 哪些代码，这样参与者可以抽出时间提前了解相关代码，对不懂的地方做个笔记以便在 Code Review 进行中提出疑问。在我们碰到比较棘手的 BUG 没有什么思路或大惑不解时，这时找几个相关人员或对此代码也熟悉的人进行一次 Code Review，这时形式比较随意，大家可以临时提出问题，让主讲人解答，在这个过程中可能听的人并不会非常快地了解其中的详细过程，但是讲的人在这个过程中重新理了一下思路，对所写的代码被迫重新审视了一遍，在其中可能就会发现出解决问题的办法。在 Code Review 时有时代码非常多，但可以一个功能模块一个功能模块地从总体到局部，由浅入深层层递进的方式进行。一次 Code Review 的时间不要太长，但可以分多次进行。Code Review 中大家会提出问题和建议，集思广益，多个人共同出主意，有些可能一个人没有想到的问题会被大家发现，互相学习，共同进步。

### 10. 沟通与交流

大部分员工的大部分时间是在公司里度过的，因此公司的生活成了大家主要组成部分。员工之间关系的融洽，交流的畅通显得非常重要，同时大家也不想自己的生活这样枯燥乏味，一直同机器打交道。沟通无处不在，交流随时发生，有许多关系是在工作之外建立起来的。软件公司内是很容易产生各种矛盾的，因为这是由你的工作性质所决定的，比如 QA 或用户会对你的实现不满意，提出各种要求时，我相信你有时会

有所抱怨的，无形之中就产生了对立，发展到后来会有抵触心理。我相信大部分人都会有此感受，这不是你的错，这主要是由我们的工作性质决定的。如果你的工作是把财富带给对方，则对方会非常欢迎你的到来，把你奉为财神爷来对待，同你的关系会非常融洽友好。因此我们需要在工作之外来消除这种对立矛盾的关系，建立一种融洽的工作氛围。我们在平时吃饭的时候饭桌上大家互相聊天沟通。我们建立了 happy 邮件列表，其中会发一些幽默笑话之类的邮件，给我们紧张的工作增加点轻松的氛围。在下班后大家可以组织一下活动，增加了公司的凝聚力。一个产品发布后组织一下旅游，让绷紧的神经松弛一下，更好地迎接下一个挑战。