





# A++ 敏捷开发

– 从个人提升到以数据说话



*impossible* publisher 2022  
printed blindfolded  
design: L<sup>A</sup>T<sub>E</sub>X



# Contents

<b>I 案例分析</b>	<b>11</b>
<b>1 如何改善</b>	<b>13</b>
某软件开发公司敏捷开发过程改进案例 . . . . .	13
如何改善 . . . . .	15
丰田故事 . . . . .	16
水面下的管理思路 . . . . .	19
现代汽车生产 . . . . .	25
改善质量 . . . . .	26
结束语 . . . . .	27
<b>II 团队与自我改善基本功</b>	<b>29</b>
<b>2 改进从团队开始</b>	<b>33</b>
美国费城 Weisbord 故事 . . . . .	33
Weisbord 故事的启发 . . . . .	35
破冰 - 变革 - 稳定 . . . . .	37
成功要素 . . . . .	37
结束语 . . . . .	39
参考 References . . . . .	40
<b>3 克服拖延症</b>	<b>41</b>
结束语 . . . . .	45
附件 . . . . .	45
5S 法 . . . . .	45
锻炼之道 The Truth about EXERCISE . . . . .	45
参考 References . . . . .	46
<b>III 自我管理开发过程</b>	<b>47</b>
<b>4 三点估算</b>	<b>51</b>
<b>5 量化管理从个人开始</b>	<b>53</b>

<b>6 估算软件规模</b>	<b>55</b>
<b>7 估算工作量</b>	<b>57</b>
<b>IV 开始过程改进之旅</b>	<b>59</b>
<b>8 获取高层支持</b>	<b>63</b>
第一版方案书 . . . . .	64
第二版与中层确认，然后向高层汇报 . . . . .	64
<b>9 寻找改进机会</b>	<b>69</b>
<b>V 如何做好迭代回顾</b>	<b>71</b>
<b>10 根因分析</b>	<b>75</b>
分析事故根因：上集 . . . . .	75
香港南丫海难 . . . . .	76
根因分析的主要元素 . . . . .	79
技术总监经验之谈 . . . . .	81
结束语 . . . . .	82
附件 . . . . .	82
二八原则与数据分类 . . . . .	82
大野耐一先生“5 Why”实例 . . . . .	83
<b>11 如何降低软件开发质量成本</b>	<b>85</b>
数据收集 . . . . .	85
自动化统计分析 . . . . .	85
数据收集频率 . . . . .	87
哪里最耗工作量 . . . . .	88
分析缺陷排除率降低质量成本 . . . . .	89
结束语 . . . . .	89
附件 . . . . .	90
质量成本 COQ (Cost of Quality) . . . . .	90
公司高层不一定关注质量 . . . . .	91
开发项目工作量（成本）分布 . . . . .	91
<b>12 如何做好准备</b>	<b>93</b>
分析事故根因：下集 . . . . .	93
回顾流程 . . . . .	95
做好回顾 4 原则 . . . . .	95
团队迭代回顾根因分析常见问题 . . . . .	98
培训：策划与准备 . . . . .	100
使用二八原则与 KJ 分析根因实例 . . . . .	101
经验教训 . . . . .	104
结束语 . . . . .	104

附件 . . . . .	104
游戏: 应与否 . . . . .	104
如何分析缺陷数据得出预测模型参数, 出分布图 . . . . .	106
Asch 1951 群众压力实验 . . . . .	110
Brehm 1956 决策影响实验 . . . . .	112
粮食分配实验 . . . . .	112
KJ 分析方法步骤 . . . . .	113
水晶球蒙特卡洛预测模型 . . . . .	115
<b>13 实施与维护</b>	<b>117</b>
准备迭代回顾 Q&A . . . . .	117
建立团队标杆 (基线) . . . . .	125
从迭代复盘到持续改进 . . . . .	125
促进公司过程改进 . . . . .	126
结束语 . . . . .	128
不仅仅适用于分析迭代缺陷数据 . . . . .	129
附件 . . . . .	131
FMEA 实例 . . . . .	131
功能性分组帮助团队成长实例 . . . . .	134
GQM . . . . .	135
参考 References . . . . .	135
<b>VI 代码质量改进的基础</b>	<b>137</b>
<b>14 测试驱动开发</b>	<b>141</b>
测试驱动开发 . . . . .	141
为什么要测试驱动开发 . . . . .	141
每小步验证 . . . . .	142
用 TDD 开发程序 . . . . .	143
单元测试与 TDD 的好处 . . . . .	144
结束语 . . . . .	145
附件 . . . . .	145
从 MIT OCW 学写程序 . . . . .	145
<b>15 代码重构</b>	<b>149</b>
如何学好 SOLID, 打好基础做重构 . . . . .	149
如何提升学员的兴趣与动力 . . . . .	150
什么时候做重构 . . . . .	150
代码规范 . . . . .	151
持续集成 . . . . .	151
附件 . . . . .	151
小孩利用乌龟玩几何游戏 (Turtle Geometry) 的例子 . . . . .	151
温伯格的教学实验 . . . . .	158
参考 References . . . . .	158

<b>16 持续集成</b>	<b>159</b>
验收测试 . . . . .	159
测试自动化 . . . . .	160
持续集成 . . . . .	161
团队协作 . . . . .	162
持续交付 . . . . .	163
附件 . . . . .	163
持续集成 . . . . .	163
技术债务例子 . . . . .	164
参考 References . . . . .	165
<b>17 结对编程与同行评审</b>	<b>167</b>
培训 . . . . .	168
总结 . . . . .	170
附件 . . . . .	171
代码可读性例子 . . . . .	171
<b>18 软件需求</b>	<b>173</b>
附件 . . . . .	175
英国煤矿生产问题的研究 . . . . .	175
<b>19 客户参与</b>	<b>179</b>
常见问题 . . . . .	179
识别利益相关者 . . . . .	180
用户故事卡 . . . . .	180
用例与场景 . . . . .	181
举例 . . . . .	183
附件 . . . . .	184
利益相关者 . . . . .	184
客户声音: Kano Diagram . . . . .	186
参考 References . . . . .	187
<b>20 团队协作</b>	<b>189</b>
各自负责自己开发的模块 . . . . .	190
团队合作 . . . . .	192
按时上下班 . . . . .	192
什么导致系统崩溃 . . . . .	193
技术团队的持续性 . . . . .	194
驱动团队改进 . . . . .	194
保持团队稳定 . . . . .	195
附件 . . . . .	196
平衡心态 Cognitive Dissonance . . . . .	196
参考 References . . . . .	196

<b>CONTENTS</b>	<b>11</b>
<b>VII 度量与分析</b>	<b>197</b>
<b>21 做问卷调查</b>	<b>201</b>
<b>22 教小孩统计分析</b>	<b>203</b>
<b>23 假设检验</b>	<b>205</b>
<b>24 控制图</b>	<b>207</b>
建立基线 . . . . .	208
噪音还是信号 . . . . .	208
控制图的应用 . . . . .	211
总结 . . . . .	213
附件 . . . . .	213
如何画 ImR 控制图 . . . . .	213
各种控制图 . . . . .	214
过程能力 (Process Capability) . . . . .	215
细分例子 . . . . .	216
参考 References . . . . .	217
<b>VIII 总结</b>	<b>219</b>
<b>25 创新: 从个人到公司</b>	<b>223</b>
引言 . . . . .	223
400 年前的重大数学发明 . . . . .	223
创新 Creativity . . . . .	225
创造的精神 Spirit of Creating . . . . .	225
公司如何创新 . . . . .	227
音乐播放器 . . . . .	227
iTunes 网上商店 . . . . .	229
公司如何创新 . . . . .	229
万事起头难 . . . . .	230
从快破产变为投资回报最好的投资 . . . . .	231
公司创新成功要素 . . . . .	232
结束语 . . . . .	235
个人经验教训 . . . . .	238
附件 . . . . .	239
Pixar 制作第一部电脑动漫: 玩具总动员 (Toy Story) . . . . .	239
参考 References . . . . .	240
<b>26 根与翼</b>	<b>241</b>
根 . . . . .	241
翼 . . . . .	243
总结 . . . . .	245

<b>IX 附录</b>	<b>247</b>
<b>27 A: 绅士俱乐部过程改进</b>	<b>249</b>
绅士俱乐部过程改进 . . . . .	249
参考 References . . . . .	255
<b>28 B: 分析迭代客户满意度调查数据</b>	<b>257</b>
案例背景 . . . . .	257
迭代数据 . . . . .	257
数据分析 . . . . .	257
改进措施 . . . . .	258
改进效果 . . . . .	258
<b>29 C: 简化功能点 (SiFP) 简介与实例</b>	<b>259</b>

# 前言 Prologue

We are uncovering better ways of developing software by doing it  
and helping others do it.

—Agile Software Development Manifesto

敏捷宣言已经有超过二十年的历史，国内越来越多软件开发团队开始采用敏捷和迭代，但总体效果参差不齐。有些年轻的团队成员听到敏捷不需要文档，以为也不需要注重代码质量，包括代码可读性，导致后面发布的软件产品问题多多，难以维护。要编写出高质量的代码，人本身能力非常关键，但软件工程快速发展，导致编程员人数快速增长，其中很多缺乏专业工程师素养，所以若要敏捷开发真正起作用，必须先提升编程员能力。

没有数据就无法管理，但很多敏捷团队只是走流程（每天站立会议、看板并非敏捷的重点），缺乏度量，所以管理者一听到敏捷就觉得不靠谱，要求团队用回传统的瀑布开发方式。这二十年来，中外都出版了很多关于敏捷的书，但绝大部分都没有深入探索以上的问题。这本书就是希望通过解读各种敏捷最佳实践，加上敏捷以外的其他知识，帮助大家理解并更好使用敏捷，提升软件的质量和总生产率，让团队成员与公司管理层获得双赢。

同时也希望管理层通过这本书能了解敏捷开发的要素，并能使用敏捷开发模式，帮助公司提升软件产品质量，同时降低成本，增加公司的竞争力。

某技术总监陈总在 5 天差距分析的最终总结会里问开发团队：

1. 你们知道自己每天产出多少代码吗？（生产率）
2. 你们知道平均修复一个系统测试的缺陷花费多少工作量（人时）？
3. 有没有发生过代码开发完成，系统测试人员尝试运行，但跑不动的情况？（开发人员不能借口说不懂业务，或需求不清，因为你写完代码后，自己连最基本确保代码能运行都没有去做。我还没问你们做开发的有没有做好单元测试/集成测试。）
4. 你们做测试的知道测试覆盖率是多少？

陈总有丰富开发经验，他最后总结说：“我以前自己做开发时也是被问过这些问题，我当时也不懂，但我能知耻而后勇——先知道现在的不足，然后按评估老师的最佳实践，持续改进。”

- 如果你像陈总下面团队一样，不懂以上问题的答案，这手册可以帮你自己找答案。

- 如果你（是管理者或客户/用户）不满意团队的软件开发成果，这手册可以帮助你找出根因，启发团队改善。
- 如果你已经很了解，恭喜你，但应该还可能从这手册里找到一些你未知但有用的方法。

1980 年，STROUSTRUP 先生不满意当时很流行的 C 语言，因为在 70 年代，已经有如 SmallTalk 之类的面向对象语言，C 是基于传统步骤式的语言，没有面向对象的功能，所以 STROUSTRUP 先生就在 C 的基础上，加上有类功能的 C (C with Classes)。过了 4 年，他同事觉得这个名字不好，就改成 C++，也出了一本书叫《C++》，与本来经典的 C 很相似。后面 C++ 就成为了面向对象的常用语言。  
A++ 继承 40 年前 C++ 的思路，希望在敏捷 (Agile) 开发的基础上，加上增值的元素，帮助团队与管理者做好软件开发，让客户满意。

# **Part I**

## 案例分析



# Chapter 1

## 如何改善

### 某软件开发公司敏捷开发过程改进案例

5 月

张工是公司的中层管理，管理好几个开发团队，有五位项目经理向他汇报。他听说老同学的团队都开始用敏捷开发，很感兴趣，并参加了一些敏捷的交流会，觉得可以解决很多传统瀑布性开发的不足，尤其是可以快速交付给客户。他要求部门经理全面推动敏捷开发，对团队成员进行相关培训，例如，SCRUM Master 内部培训。

开始时，张工上级部门经理有些怀疑，问：“后面那些工程文档都不做，会不会影响质量和交付？客户都是专门做电信的，不缺钱，但是对质量要求很高。”张工便解释说，“只要利用敏捷把过程变成迭代，快速交付，改善工程的问题不难，主要是人的问题。”

部门经理听到敏捷可以比以前更快速交付，之前客户经常因为延误而不满，他希望可以改变这现状，就答应了。

8 月

开发组长王工周五下班后与朋友喝酒，开开心心说：“太兴奋了。研发部门经理决定全面推动敏捷开发 SCRUM；我们团队刚参加了两天培训，真正对应我们以前的传统瀑布式开发的种种问题，我们会 2 周一次迭代，快速反馈，我们会定期小步向客户发版，我们会与用户经常交流，获得他们的反馈。

现在团队合作不像以前只按工种工作，也会跟产品经理、业务方面更充分合作，给客户带来更高价值。

工作方式也改变了，以前要写需求、规格说明书，现在简单化成用户故事和产品需求卡片，以前我们要做过详细项目计划甘特图，现在用改成用燃烧图和白板。每天用便利贴去写要开发什么东西贴在白板上面，开始的时候，贴的越多感觉越敏捷，我们改成叫 SCRUM team，有一系列的海报围绕我们。我们也没项目经理了，自己管理自己。本来的部门经理现在变成产品负责人，敏捷开发方式让我们团队自己做决策，不仅仅是技术方面，项目相关的也由我们项目组

一起讨论决定。

解除了以前‘瀑布式开发’的种种烦恼，这一切太完美了。“

### 9月

”你们团队学完敏捷 SCRUM 后，项目如何？”

王工充满自豪地说：

“我们培训后就 SCRUM 的方法，定每两周一个冲刺，每次冲刺前都会用故事点

来估算每个功能多大，然后按本次冲刺的资源，估计可以完成多少功能？

然后用白板来监控模块完成的情况，哪些在开发中，哪些已经完成，团队和管理者都可以一目了然，不用像以前天天问我们了。我们每天早上也按照 SCRUM 的规定站立会议，每人都说自己完成了哪些任务，今天做什么。

大家都很兴奋，确实跟以前瀑布的做法不同。”

### 10月

”你们项目如何？”

王工听完，想了一下，然后说：

“我们本应上周要完成一次冲刺后的割接上线，但被推到下次了。”

”为什么？”

王工说：“我们按培训学到的做冲刺计划会，按照产品的待办事项列表，团队利用扑克牌一起估算每一事项所需要的时间。我们总共八位开发人员，其中有一半是刚毕业不久，但大家刚上完培训，很有信心，虽然技术主管张工对我们出来的估算有些顾虑，觉得我们太理想，但大家刚培训完敏捷，张工也希望让部门经理尽快看到敏捷开发可以加快速度，我们就按这‘进取式’估算开展 2 周冲刺。但因新人多，编码水平有限，虽然大家已经尽快把开发出来的代码交给系统测试人员手工测试，依据测试发现的缺陷修正再测试，但越来越接近答应客户的 2 周割接上线时间，但是还是很多 BUG 没改好，最后几天，基本就天天加班，最终到验收时，仍然有不少问题，最终割接前测试，还是不能达到客户要求的水平，没办法，未能上线。

大家确实都尽力冲刺了，但未能达到我们本来希望的结果。”

### 11月

部门经理之前收到客户总监电话，投诉一些技术缺陷，导致好几次不能按计划上线，问为什么正在交付的软件质量变差了？

张工被问到是什么原因时无法回答，只能说立马回去探索原因，尽快汇报，但心里想：“开始敏捷后，因为快速迭代，以前要做概要设计、详细设计的过程反而被忽略了，导致有些写出来的代码，后面就很困难适应快速的变化修改，导致要不然就功能做不出来。因要赶时间，可以按客户的要求时间交付的话，由于本身代码不好，只可以临时凑凑，不长久。”

张工从部门经理办公室出来后，找其中一位项目经理李工喝茶，回顾一下发现项目团队对这次敏捷 SCRUM 的改革有意见。例如上层为了更快速交付，实现敏捷可以快速交付承诺，把一些本来不太可能的进度时间压到团队去，完全不是本来的那种自主团队管理的概念。出现问题多了，就请了敏捷教练过来辅导，但 SCRUM 的教练也缺乏软件工程的基础，只懂项目管理过程。所以他们也解

决不了软件相关的问题。只是把精益管理怎么做迭代，怎么做回顾这些基本过程再解读一下，解决不了实际问题。

李工：

“因为我们做这块业务已经很多年了，本来业务的变化不多，只是一些小的功能改动，所以开发人员尽量不去动核心的代码，怕改动了反而会影响投产，切割不了。但有时为了满足一些新功能，继续在老代码的基础上去写，这种做法效率很低，也不长久，估计一两年后会难以运行了，我们会被迫重写整个产品，而老代码开发人员大部分都离开了，后面的代码维护变得非常困难，即使用敏捷也解决不了这个问题。”

无论张工或李工也没有能去总结出什么好的解决方案。现在推行敏捷才刚刚三个月，绝不能打退堂鼓，回到本来的状态。但应怎么解决敏捷带来的问题？挽回部门经理与客户的信心呢？

## 如何改善

从以上案例看到，本来管理层希望利用敏捷开发，加快软件开发的交付，减少延误，令客户更满意。但因为只注重项目管理，但没注意和改善软件开发本身的质量，人员能力等因素，开发出来的软件缺陷比以往多，导致后面大量返工，恶性循环，后面更导致延误，最终客户投诉。

因为软件本身设计有问题，导致软件难以修改，开发人员都不敢改动任何代码，怕可能会引起系统崩溃。

怎样可以确保开发出来软件的质量？

敏捷开发有很多种方法 (SCRUM 只是其一)，因为目的不仅仅是管好项目进度，也要确保软件产品的质量。所以 SCRUM 只包括项目管理部分，不全面，反过来，例如极限编程 (XP eXtreme Programming)，因它的发明者 Kent BECK 本身是一位精通面向对象的程序员，所以 XP 不仅仅关注项目管理，也包含编程的最佳实践。下图是 Ron Jeffery 把 XP 的重点画成从外到内 3 层：

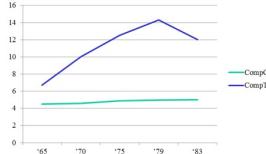


SCRUM 只包含了外层的部分，缺乏中间和内层元素。按 XP 的 12 实践（详见附件）都做到了便可以解决张工的问题吗？

先问你以下关于汽车公司的问题：

### CompT vs CompG (T 与 G 两家汽车公司)

- 1955年前后, 生产一辆汽车的成本: CompT 成本 = CompG 成本 x 2
- 全球销售额: CompG 销售额 = CompT 销售额 x 140
- 平均每员工年汽车生产量



T 公司在五六十年代, 在销售, 生产成本都远远不如 G 公司, 但它每年生产率都一直提升。

请猜猜 T 和 G 是那家公司? (提示: 两家都是世界级汽车公司, 现在你还能买到它们生产的汽车。)

=====

G 是美国通用; T 是日本丰田。

为什么丰田能从一家战后小公司能提升为世界最大的汽车公司?

### 丰田故事

200px

丰田喜一郎先生 (创始人)

200px

二战后 50 年代, 丰田汽车规模很小, 经营很困难, 在破产边缘。但丰田创始人喜一郎先生明白美国生产线的弊端, 意识到未来的汽车生产必须是 Just-In-Time::

- 每一辆都是按客人订单订制: 例如, 颜色, 配置, 左右钛等
- 从钢材原料开始, 整个生产线, 零等待, 零浪费

每个工作步骤所需配件按生产需要到达 (不晚, 也不早到), 把生产过程中的配件降到零



\* 0 defect 零缺陷

100% value added 百分百的增值

One-piece flow, in sequence, on demand

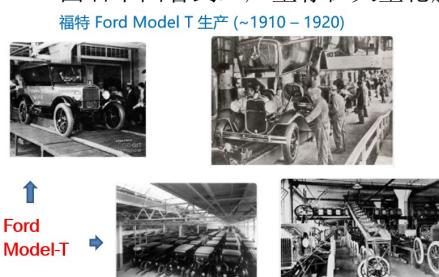
按订单的流水线生产，过程中零报废

喜一郎先生安排总工大野耐一去美国汽车公司考察。

为什么福特 Model-T 出名？它是第一辆一般人买得起的汽车。首辆现代汽车是由德国奔驰 (Benz) 于 1885 制作，但是这些第一代汽车都非常昂贵。老福特是一名工程师，对机械特别感兴趣，白天是爱迪生公司的总工程师，到了下班后就研究汽车发动机经过十年的不断优化，最终开发出 4 冲程 (Quadcycle, a self-propelled) 自动汽车发动机。他的发明大大降低了汽车组件的成本。1908 年，Model-T 首次推出市场，售价是 850 美金，虽然已经比同期的其他厂家汽车便宜，但还是超出一般工人家庭可以负担的水平。

为了进一步降低成本，老福特觉得必须想其他办法。汽车组装一直都是作坊式运作，所以组装汽车需要超过 12 个工时。为了降低组装时间，必须要学其他工厂，如啤酒，麦粉厂，它们都已经是流水式生产线生产 (Assembly Line)。福特把整个 Model-T 组装拆分成 84 步骤，并培训工人只做一项工作，还聘用了科学工程师 TAYLOR 先生设计整个流程，提高效率，最终可以在一个半小时完成组装一辆汽车。生产线生产帮助福特公司把成本和售价降下来，在 15 年期间 (1913 - 1927)，福特公司总共生产了 15,000,000 辆 Model-T。虽然大量生产能直接降低生产成本，但这种做法也要付出代价：

- 为了生产所有都是同一个款式同一个颜色，黑色
- 为了提高生产的效率，也增加了组件的库存量（为了提高效率，降低成本，大部分组件都要批量生产 (Batch production)，例如轮胎。例如上面右下图看到工厂里存在大量轮胎。）



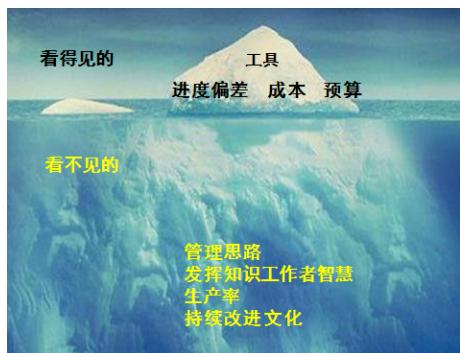
Federick TAYLOR 与科学管理 (Scientific Management)



十九世纪末美国很多工业快速扩张，Taylor 先生发现当时很多工厂生产力很低，例如钢铁厂。他是很落地、实干的人，他就想有什么好方法可以让提高生产力，工人也多赚些钱。他发现很多工作都未设计好，未能好好利用工人，他针对工作本身做很多科学研究，测量每个工作应该多长时间，怎么做最好，然后按他的最佳设计把工作细分，也同时要求公司提高员工的奖励，希望工人看到因科学管理，提高生产，个人也受益。1890 年，他加入了 Bethlehem Steel，钢铁业的大公司，帮它设计了很多奖励制度，也设定了一些叫工业工

总工大野耐一先生，从美国的超市（非汽车公司）得到如何做 Just-In-Time 的启发，回国后就开始在丰田致力推动。开始时，很多人都觉得 Just-In-Time 这个愿景好像是远不可及的梦想。

## 水面下的管理思路



为什么丰田能成功地把汽车生产做到 Just-In-Time，超越西方的巨头，使日本汽车制造过程成为世界“标准”？但我们不能单从表面看这“系统”的方法和技巧，例如大家都熟悉的看板管理（他的竞争对手通用、福特、肯定都学过），更重要是了解背后的管理思路。我们看看下面丰田九条原则，开始了解水面下的“系统”：

### # 以忙碌为耻

1. 培养人才 - 逼他们动脑筋
2. 从心里相信“大家的力量”
3. 不以“我们公司”作主语
4. 标杆管理 (Benchmarking)
5. 容易实现的目标不是好目标
6. 根因分析
7. 成长比成功更重要
8. 所有人参与改进

### 以忙碌为耻

- 不吝惜智慧，但要吝惜汗水

把“动作”转变为“工作”以前，当大家批评某机关的工资太高时，职员会以上班时间很长，“一直在努力工作”为由来反驳人们的批评。这其实是一种对于“有动作”和“在工作”的混淆。不管上班时间有多长，如果没能够创造出利润，那么就不能称之为工作，也不能称之为一直在努力。丰田是把“动作”和“工作”分开考虑的。

在丰田看来，“就算一直在动也不代表那个人在工作”。省掉徒劳动作，把“在动着”转化为“在工作”。

大野耐一先生曾经问过年轻员工：“每天工作一小时左右，你们能做到吗？”听了这句话后，有人抱怨“算上加班时间，我们一天工作九个小时，他那句话是什么意思？”确实，他们在公司待了很长时间，但如果把“有动作”和“在工作”分开考虑，九个小时中，真正在工作的时间可能只有一小时。大野先生指的是这一点。

关键的不在于流着汗在公司转了多长时间，而要把自己的工作区分成“徒劳作业”和“有价值作业”。

因缺乏数据，很难判断这些软件开发人员每天有效生产多少？什么因素妨碍员工生产率提升？

{| class="wikitable" | 二战后，日本经济萎缩，丰田辞退了不少员工；1950年，朝鲜战争，需要大量增产，但丰田选择了只增加设备，而不增加人手，大野先生也借这机会，完善丰田生产方式，成功找到了以不增加人手为前提的增产办法。

|}

### 培养人才 - 逼他们动脑筋

人了不起的智慧 所谓丰田生产方式其实就是要建立起一种体制，把“人了不起的智慧”引导出来，使得这些智慧能在生产一线得到充分发挥。所以说“丰田生产方式源自人的智慧”。

企业相信员工的智慧以后，员工们的干劲、使命感、责任感都会随之而生，最重要的是，员工们会因此逐渐对工作抱有自豪感。一旦员工们的意识改变了，理所当然地，企业的竞争力也将获得很大的提高。

丰田生产方式是 TQM (Total Quality Management 全面质量管理) 的最佳例子，TQM 强调专注客户、持续改进、以数据说话、员工参与等，丰田生产方式覆盖了 TQM 原则的七项 (除了战略)



从心里相信“大家的力量”

- 不是靠“一个不平凡的人”，而是依靠“一百个平凡人”来创造亮眼的成绩
- 生产产品就是培养人才

“做事业最关键的是人……‘培养人才’为基础”总裁丰田英二先生

有一位丰田员工被问到“丰田生产方式到底是什么”的时候，这样回答：“就是在人的智慧建起的基础上，立起了自动化和及时生产这两根支柱。”

他所说的“人的智慧”是指“在一线工作人员的智慧”。

不以“我们公司”作主语

- 不是从“专业”的角度，而是从“顾客”的角度生产产品

创业大忌 - 闭门造车，所以丰田的原则，对客户有用就一定做出来，但对客户无用，或者不想要，就绝不生产。



要创新必须从“顾客”的角度看，不单从工程师的视角看不仅适用于丰田和汽车制造：

征服太空，踏上月球六七十年代，美国开始阿波罗太空计划。首批宇航员强烈要求工程师加上窗户，逃生门，可人手控制等（现代我们会觉得这些是太空船 (spacecraft) 基本要求，但当时是闻所未闻）  
 1980，一位 NASA 工程师回顾当时工程部的想法：“你们又不是火箭专家，航天专家，只是飞行员。希望可以在火箭驾驶舱人工控制火箭飞行，开玩笑！太极端，你们可不了解成本多高，我们应可以很轻易用工程的理由拒绝。”

### 标杆管理 (Benchmarking)

丰田（如下图）很注重各种标杆，例如内部标杆、竞争性标杆等。软件开发也应该同样利用数据来制定量化目标，例如生产率。最近行业也要求做功能点估算来统一软件报价，有了功能点，团队就可以利用它来衡量软件团队的生产率，它也有不同行业的标杆做参考。



容易实现的目标不是好目标

- 不是“削减一成”，而是通过“取消一个零”来发现浪费

把原先需要三小时的工作，改用三分钟完成。

听到上司这个要求，你该怎么回答呢？多数人会常识性地回答：“这太强人所难了”、“绝对做不到”。

## 丰田

单分换模 (Single Minute Exchange of Die)" 例子:

1965 年, 丰田汽车在推行丰田生产方式时遭遇一个瓶颈: 装置更换时间太长, 其中特别是 500 吨冲压机和 1000 吨冲压机的模具, 更换时间长达 4 小时。如果不缩短这两个模具的更换时间, 就不可能实现多品种少量生产方式。

挑战由大野耐一先生统领, 在生产管理的先行者, 新乡重夫先生的指导下, 分两个阶段展开。

改善开始之前, 新乡先生凭借自己多年的工作经验, 了解到装置更换有两种方式。

内部装置更换——必须在机器停下来以后, 才能进行的装置更换。

外部装置更换——能在机器运转过程中, 或是在运转起来以后, 进行的装置更换。

要缩短时间, 把内部装置更换和外部装置更换清楚地分开来是一个关键。能在外部装置替换作业中进行的工作, 就全部在外部装置替换过程中实施。同时, 分别对内部装置替换和外部装置替换进行改善。通过这种方法, 装置更换时间缩短为一个半小时。

完成这一改善花费了半年时间。通常能有这样的成果就可以告一段落了。但是, 大野先生仍要求进一步缩短时间。

他要求把更换过程“缩短为 3 分钟!”通常通过改善能把原先的 2~4 小时缩短为一个半小时, 就可以很满意地说“已经很好了”。但是, 大野先生不这么想。他认为: “既然能缩短到这个水平, 那么继续改善肯定可以把时间缩短为 3 分钟。”

改变汽车生产的单分换模的秘密

对于这一要求, 在以新乡先生为中心的技术小组中, 自然有人提出“3 分钟绝对干不完”。但是, 新乡先生认为: “如果能把内部装置更换全部转化为外部装置更换, 3 分钟也不是不可能

之后, 他着手对多达 100 个以上的项目进行了改善。

首先, 进一步对内部装置更换和外部装置更换进行细分, 彻底把内部装置更换转化为外部装置更换。同时, 想方设法对各种切割工具和模具进行设置, 使得更换时用一个动作即可完成。此外, 在紧固件上也动了很多脑筋。这样, 终于创造出了无数项不花时间、能够简单完成同时可以在作业时保持稳定的改善。

紧接着, 对作业顺序反复进行改善, 实施标准化 (制定没有多余工序的作业标准)。这样, 在挑战进行了三个月后的某一天, 真的只要 3 分钟就能完成了! 这个结果让所有人都大吃一惊。

## 根因分析

(五个为什么是一种找根因的方法, 详见附件 B)

- 不停留在“原因”上, 而要找出“真因”, 彻底改善
- 不追究责任, 而是追究原因

丰田的大野耐一先生说:

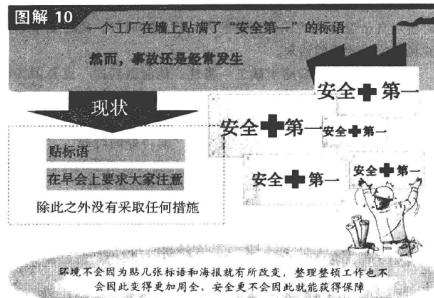
做到一半是不行的，只有一个期限，“到完成为止”

当工程师报了有问题就必须要求工程师查找原因、提供数据，但很多时候这个工作并不简单，但大野耐一先生决不放弃，必须找到根因。

### 成长比成功更重要

- 要培养人才，“改变体制”比“改变人”更有效

有些工厂只依赖张贴标语、海报，希望可以减少工地的事故发生率，但丰田不注重喊口号，而是动手干实事，包括机械保安、设备保安等



叫人做之前自己先做一遍给他们看 本田技研工业在中国创立广州本田的时候，派往当地的都是即将退休的經驗丰富的中高年龄层技术人员，这批技术人员成为了厂里的中坚力量。正因为他们的努力，广州的本田工程才获得了成功。

## 所有人参与改进

带诚意去赢得协作 B 先生所在的 A 公司曾以丰田生产方式为基础进行了生产改革。

要进行生产改革没有技术部门的配合是行不通的。但是,任凭 B 先生怎么要求,技术部门依然毫不合作。B 先生实在没有办法了,只能向总裁要求:“请加大我手上的权力,让我可以支配技术部。”总裁回答:“你去给我请教了大野(耐一)先生以后再说!”

于是 B 先生去找大野先生,在听他诉说了自己面临的窘境以后,大野先生对他说:“你这一两天跟我一起去工厂转转吧!”并于百忙之中抽出时间带 B 先生参观了丰田的工厂以及附近的协作企业的工厂。这期间,大野先生什么话都没说,只在第二天下午问 B 先生:“怎么样,你明白了吗?”

B 先生回答:“我觉得在工厂听到的关于厂长的改善事例,跟丰田方式所强调的重点好像不太一致。”听了 B 先生的回答以后,大野先生点头:“就连我,也是一直都在忍耐的啊。工作并不是有权力就能解决问题的。要想得到对方的理解和信任,拿出诚意去找人家吧。”

从那以后,B 先生再也不找“因为我手里没有权力”之类的借口,而总是带着诚意去找对方协商,不久以后,他成功地对 A 公司实施了生产改革。

每当听到有人感慨“下属不听话”的时候,一位曾在丰田工作的人就会说:“你要求自己的孩子”每天学习三小时’时,他会听话去学习吗?”

对方的回答是:“估计没用。”

“连自己的孩子都这样,更何况那些成年的员工呢?”

## 现代汽车生产

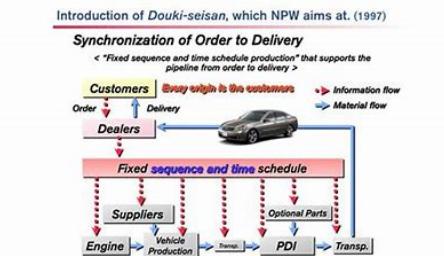
今天 Just-In-Time 已成为汽车制造的主流,例如:日产在英国牛津 (Oxfordshire) 专门生产 mini 车的工厂便能做到:

- 从钢材原料到生产出汽车只需要 24 小时
- 整个生产线没有任何中间等候, 每 68 秒出一辆
- 每天生产 1000 辆车



挑战: 每一辆汽车都不同 – 颜色, 设备, 左右驾驶座等

- 生产线上每一辆汽车都按照客户需求订制
- 组件不早不晚按需求准时到达生产线
- 这便需要信息化系统把客户订单转换成生产信息



例如，生产线上每一辆的颜色都可以不同：



## 改善质量

乔布斯离开苹果公司后，自己开创 NEXT，他接受访谈时总结了质量改进的重点：

质量提升的道理其实很简单，是重复的过程：我们需要不断去看，有哪些无效的环节要省略，哪部分要重新设计，不断试验、提升，就这么简单。重点是所有的提升都应该是科学化的，有数据而不是泛泛而谈。  
一般管理层的思路是：我是领袖，你们应该听我命令。但应该是反过来，让应如何做好的决定权利放在团队手上，做改进不需要请求管理层的批准。改进是工作的一部分，整个架构扁平化，自己管控日常过程，每位工程师应像以前的工匠，愿意花精力不断做好。然后能以自己最后做出的优质工艺、产物自豪。

要了解质量改进，先要了解质量的定义：

按质量大师裘兰博士 (Dr JURAN) 定义，质量包括两部分：

- 满足客户需求（客户包括外部客户和内部客户）
- 没有缺陷

这定义不仅仅适合于制作业、也适用于服务业，IT 业。

质量管理，与财务管理类似，同样也有质量策划、质量控制和质量改进，基于这大框架再细分：如何制定质量目标，如何来制定度量等。

质量策划包括：

1. 设定目标，包括外部和内部目标
2. 识别内部需求
3. 依据客户需求制定产品的功能特征
4. 制定产品和过程的目标
5. 设计过程来达到这些目标，最后验证过程能力

所以如果希望利用敏捷开发，不仅仅是走迭代，确保进度没偏差，还要确保软件产品质量，也应该用裘兰博士的质量管理思路去看敏捷过程，才能更全面了解如何才可以确保软件开发的质量，也控制好交付工期，不延误，让客户满意。最佳实践，如果没有定质量目标，配上度量衡量和策划，都只是空想，对长远提升公司文化、团队成员的习惯没有任何帮助。举例：参照上图，例如我们想改善团队的策划和估算。首先要识别客户，哪些是主要干系人 - 甲方有什么要求；内部部门经理，有什么要求。然后从他们的诉求变成过程的功能和特征，但如果特征只是描述，没有数字也没有意义，所以要配合可衡量的度量单位，和用什么方式去收集那些数字，然后依据目标定过程要怎么去做，怎样改。

不是单纯‘空降’某敏捷流程（例如 SCRUM），便能加快团队的发布速度。所以虽然极限编程里每一条实践都是最佳实践，也必须配合质量策划，和监控改进才会有效果。

## 结束语

从丰田故事看到“系统”如何帮公司培养知识工作者，发挥人的无限智慧，为公司增值。

汽车制造大部分利用自动化机器，但当今软件开发生产和质量，非常依赖开发人员的水平，所以我们更需要建立“系统”，帮助员工快速成长。

前面分享的九个丰田管理思路：

1. 以忙碌为耻
2. 培养人才 - 逼他们动脑筋
3. 从心里相信“大家的力量”
4. 不以“我们公司”作主语
5. 标杆管理 (Benchmarking)
6. 容易实现的目标不是好目标
7. 根因分析
8. 成长比成功更重要
9. 所有人参与改进

丰田汽车从 50 年代开始源自裘兰博士的质量管理思路，成为世界最大的汽车企业。

针对软件开发，如何基于以上质量管理、精益管理思路，配合敏捷开发最佳实践，提升产品质量与团队竞争力？

所以团队成员，包括团队组长与组员，的能力是过程改进的基础，不仅仅依赖领导人。

下一部分，我们探索‘团队与自我改善基本功’与成功要素。

## **Part II**

### **团队与自我改善基本功**



改善都应从团队开始，并要改善本来的工作习惯的不足，但改变工作习惯不容易。首章用实例介绍公司团队如何从‘破冰’到变革，到稳定，和成功要素。团队的提升依赖于个人的成长与提升。敏捷开发假定每位成员都具备极高的经验与能力，如果团队成员都是刚毕业的新手，不建议使用敏捷开发。后面一章介绍提升个人效率的方法。



# Chapter 2

## 改进从团队开始

从下面两家美国公司五六十年代做改革的成功故事，让我们了解自主团队如何能带动公司提升。

但是这改进思路并非万事万能，后半部会总结背后的主要成功要素。

### 美国费城 Weisbord 故事

60 年代复印机还未普及，很昂贵，所以为各类公司客户提供印刷服务就有市场，这个故事的主人翁是某美国东岸一家印刷公司老板的儿子 Weisbord。他一直都没有接受什么正式的管理培训。他有一个好朋友 Don 在国际大公司已工作了 20 年，Don 推荐 Weisbord 说：“很多大公司已经开始推进团队自主管理，提升生产率，你可以先读 X-Y 理论（McGregor ‘Human side of Enterprise’）一书。”

X 理论：

- 对（员工）人性的假定
  - 懒散 / 不想工作，尽可能避开责任
  - 依赖/被动 - 希望被指挥，工作安稳最重要
  - 因此主管必须要把工作细分，并要紧密管控才会达到结果



Y 理论：

- 假定
  - 关注工作，希望负责任 (take responsibility)
  - 希望成长，有成就
  - 如给予机会，会尽力把事情做好

问题出于：X 思维的主管

Weisbord 本来只想看看有多厉害，但他结果一口气一周末读完全书。他发现自己到处都是 X 理论管理：

- 工作分得很细，每员工不清楚全局
- 任务都是依赖主管分配
- 员工遇到问题、困难，交给主管处理
- 员工上下班打卡

他觉得这方式应该可以帮公司提升，但他担心单靠自己推动不了这个改革，他便请 Don 过来正式加入公司。

他们分析公司业务最大问题是订单处理部门 (Order Processing)，每个订单处理工作分到五、六个任务，员工只管被分配的任务，例如：

- 筛选客户邮寄地址，邮寄印刷样本
- 输入订单
- 检查客户的信用
- 发起内部生产订单
- 用打字机打发票邮寄出去
- 收到的支票，对上那些应付账

公司平均每天要处理 200 到 300 个订单，但因为工作分得很细，如果某个任务（如输入订单）有一、两人缺席，便严重影响整个流程，效率会降低一半。

针对这问题，Weisbord 计划改革，把这部份的人分成多个 4-5 人的团队，把公司的两万个客户按地区分到各团队，每个团队有自己的客户名单，打字机等设备，希望以此增加部门的灵活度，并提高生产率。

Weisbord 与 5 位小主管 (supervisor) 商量，2 位很赞成，2 位中立，其他 1 位觉得不会成功。Weisbord 决策启动这改革，与 Don 开始重组部门。

### 开始团队制

开始时问题非常多：

- B 物流公司运送出错，送到另一个城市，怎么办？
- D 团队误解了生产的步骤，怎么办？
- C 团队的样板工人，刚入职 3 周，对我们公司产品线不熟识，怎么办？

原因很简单：本来每个人以前都只懂自己的一小块工作，以前问题都是由小主管处理。现在自主团队没有主管，都要靠团队自己想办法，像瞎子摸象。

但问题确实太多了，没办法，Don 建议每周开讨论会。他们从未有每周开会的习惯，Weisbord 本来以为开会浪费时间，把时间用于生产更实际。问题一直都非常多，延续了一个月。

Weisbord 开始怀疑这 X-Y 理论，只是大学象牙塔里的玩意，难以真正用于实际公司环境。Don 也没有更好的建议。

Weisbord 开始有撤销整个改革的想法，返回本来的组织架构，Don 还希望 Weisbord 稍等一两周，看看有没有好转。但 Weisbord 觉得一直这么多问题，严重影响公司运作，也难以与父亲交代，想在下次开会时向大家宣布变回本来架构。

到了第五周开会：

Weisbord, 与以前开会一样, 先问: 大家有什么问题?

沉默, 几分钟后, 某组长说: 没有问题。

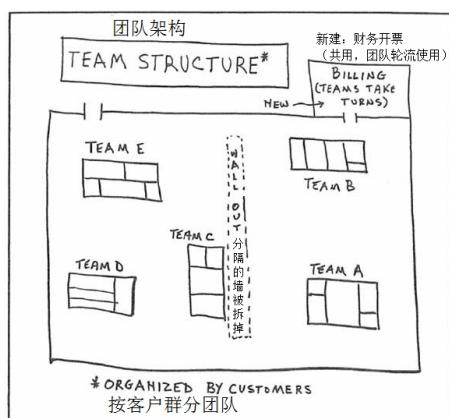
Weisbord: 为什么会有? 一直这么多问题 (心里想, 一定是你们也放弃了, 连问题都不愿提了。)

另一组长回答: 我们本周没有什么问题, 遇到的问题都在以前的会议里面被解决了。

## 效果

改革的结果是从原来每天低于 300 个订单提升到了 400 个。员工的出勤率也得到了改善, 缺席率降低到接近零。

改革后的订单处理部平面图:



(团队自主改革前, 小主管们为了减少各功能之间在处理订单引起的争吵, 特意在中间加了一道墙; 后面团队们一致建议把墙拆掉。)

Weisbord 事后回顾: 所有改变, 重组都需要时间磨合, 过程中管理者的支持非常重要, 如果管理者不能坚持, 面对并解决面对的困难, 便会以失败告终。

## Weisbord 故事的启发

如果员工具备知识技能, 管理者便应尽力给员工平台, 让团队自主发挥, 才能不断提升竞争力, 应对变化万千的市场需要, 让员工与公司都受益; 管理者不仅仅是制定目标、计划, 发号施令的角色, 而更应该是一位导师, 辅助团队成长。  
==== Lewin 教授 (心理学): “必须参与一起分析、讨论, 才有动力后面采取行动。We are likely to modify our own behavior when we participate in problem analysis and solution, and likely to carry out discussions we have helped make。”这些实验也解释了为什么虽然 Taylor 先生科学管理本应可以提升工人的效率, 但因为整个设计都是由专家设计, 工人没有参与, 所以反对。

Lewin 教授的实验与分析启发了随后的学者继续实验研究。

### 睡衣工厂实验 Harwood Manufacturing

二战后，在维京尼亚州 (Virginia)，Harwood 是一家专门做睡衣的工厂。

难题：

要按不同的季节需求，做不同款式的睡衣，但往往有些女工习惯了某种工作方式，效率很高，但是要她改变工作方式，就受不了，很难转换。

**实验 - 比较三种变更方式：**

\* 按传统方式，依赖工程师去设计每一轮的变更。团队只需要按设计去做。

- 每个团队有代表与管理层去讨论如何变更，然后执行。
- 自己去策划自己的变更，也提建议。也安排那些女工每周跟那些生产率高和低的女工一起讨论不同方式的优劣，自己讨论怎么改变。

**实验结果**

\* 第一组按传统强制要求她们改变，花了 3 个月的时间，并引起很多不满，效果也不理想：生产率降低了百分二十，也发现有十个女工里面有一个后面就辞职不做了。

- 第二组效果中等，需要大概两周才恢复到本来的水平。
- 第三组发现两天后就已经返回本来变更前的生产力水平，后面慢慢的上升提高到百分之十四。例如，某一个实验组就把本来一天 45 件，五天后提升到 87 件——一个本来无法想象的数字，后面甚至提到 90 件，超越本来那些工业工程师的目标。

这实验验证了集体讨论、自己管理，确实可以提升团队的生产率。结果也验证了实验前的发现：不同工人有自己不同的方式做事，没有像 Taylor 先生深信那种工作的最佳方式。

### 结论

问：这些故事是否想表达：我们就好比 Taylor 先生只是从工程方面去做优化。因为团队人员没有参与分析，导致他们难以接受和执行工程师（我们）设计的最佳方案。

答：你说得对。这些实验验证了群体行为：如果他们有参与或者可以影响结果，他们就有动力去做改进，如果要求他们听专家的最佳方案，往往就没有效果。五六十年代，越来越多公司采纳这个方式去做改进，也很多大学机构做这方面，培训不是用什么传统教学问的方式，而是用一些角色扮演或者团队的集体问题解决，去强调一个团队只要他们有收到及时的反馈，也比较开放，就可以做到一个民主的改进方式。下面另一个实验可以让我们更理解：

自己做实验，取数据分析一家工厂，很多主管都不愿意聘用三十岁以上的女工做机械操作工作，都觉得她们的生产力不如年轻女工高。你估计如果我们做顾问，给他一些数据，它会改变吗？不会。所以顾问只鼓励他们自己去做一些实验，证明那些三十岁以上的女工确实生产率比年轻的低。他们确实按这个做了各种实验，但分析结果发现那些年纪大的女工无论生产力率，缺勤率、学习速度，更换工作流程的速率，都比年轻的好。后面他们也不再拒绝聘用年纪大的女工了。但其他那些没有参与做实验的团队，一直都不愿意改变。

### 破冰 - 变革 - 稳定

以上两案例都可以归纳：开始团队可以归纳：自主团队管理都都需要经过下面三个阶段：

第一：破冰 (Unfreezing)，需要提供一些新的信息、新的数据，来减小/降低团队的反对声音。

第二：变革 (Moving)，要他们团队变态度、价值观、架构、行为等等。从以上的实验这个不能仅仅靠一些工程师去设计，最好是让他们自己讨论，得出他们觉得最合适的方式，他们才愿意按他们的讨论去执行。

第三：固定稳定下来 (Refreezing)。当有了提升以后，需要有一些机制去维护新的行为方式，不要让它退回本来。与团队一起去研究、分析变成行动。顾问只是提供数据去破冰，减小反对的力量。所有变更、变革都会很痛苦、很困难。顾问或者公司管理层都应该有这些心理准备。例如本来的小组长、主管，他要接受在变更后角色的变化，不像以前是像将军一样继续下命令。

也类似过程改进戴明环 (PDCA)：

Incremental Process  
Improvement 增量过程改进



25

从以上这些实验你应该也可以理解为什么本来 Taylor 先生那一套科学管理方式的不足。他只是考虑了工作方面、工程方面的因素，忽略了人性方面的考虑。但千万不要忽略工程方面的重要性。

### 成功要素

上面案例里的 Weisbord 先生，后来卖掉了父亲创建的公司，自己成立顾问公司，他从六十年代开始做了几十家美国公司的改革，也发明了六个盒子方法

(Six-Box Model)。他的经验分享：

“每次跟企业做访谈、诊断时，一般顾问单是从公司遇到什么问题、什么困难、如何解决？跳出这个圈圈，从更高的角度去想这家公司有什么改进的潜力？有哪些人在公司里面可以持续推进这种改革？有没有一些让公司每个人都愿意参与的改进方向？那现在环境变化很快，顾问的角色应该是跳出为客户解决一些技术问题，诊断给药方，而是挖掘出员工的潜力，让他们动脑筋一起思考如何解决他们遇到的问题，才是一个对企业更有价值的东西。”

他建议要注意 4 个成功要素：

### 公司潜力 Assess the Potential for Action

#### **Leadership commitment 领导支持**

这点最重要，顾问的责任不是诊断技术问题提供解决方案，更重要是探索公司有没有持续改进的潜力？如果缺乏的话，什么神医也医不了公司的病。

- 有没有领导的支持？我接触很多公司，如果高层联系不上，或者不关心质量问题，觉得都可以委托他人解决，就很难做了。最常见就是老板是业务出身，一直都没有太关注软件工程，觉得写一个软件没有什么困难，市场和销售才最重要。反过来，如果有技术总监等，甚至软件工程质量的欠缺或者重要性，兴趣听可能的一些解决方案，这个公司才有潜力做一些持续改进。你可以想象如果我们开展培训，要求他们参加，如果高层不支持，后面什么都做不出来。下面的人也知道我们顾问，或者我们讨论后有什么建议、结果，老板或者高层也不关心，肯定就不会有什么效果。

### 商机 Good Business opportunity

公司有没有一些持久的商机。在软件工程竞争很大，如果公司没有一些持久的方案、产品或者针对性。只是靠客户的关系接一些开发项目来做，就很难有一个长期的资源去投入，提升自己的质量。恰恰从我们的经验，这类软件开发公司会越来越难在这个市场上面竞争，比一些有能力、产品化的公司取代。

### 员工动力 Energized people

公司里面有没有一些中层项目经理级别的人愿意投入时间，真心推动改革，原因是顾问只可以短时间帮助公司，长远还是要靠公司内部人员去推动。怎么识别有没有那些人呢？人可以分 4 种心态：如果是在下面两格的人的话 (Denial= 否认, Confusion= 迷惑)，就不用想会支持这种变革，首先必须要是在上面的两格 (Contentment= 满足, Renewal= 更新)。所以作为顾问，你要识别有没有这种人，在公司里面要辅助他们作为整个公司改革变化的重要种子。



#### 4 种心态与对策:

心态	表现	对策
Contentment 满足	满足现状 I like it just as it is.	不打扰 Leave people alone.
Denial 否认	你说我胡说！胡说！ What? Me worry?	提问, 支持, 不能提建议 Ask Question. Give Support. No advice.
Confusion 迷惑	乱七八糟, 整个! What a Mess! Help!	振奋未来 Focus on Future. Structure tasks. Get people together.
Renewal 更新	有很多改进想法/建议 We have many good ideas.	支持执行 Offer help for implementation.

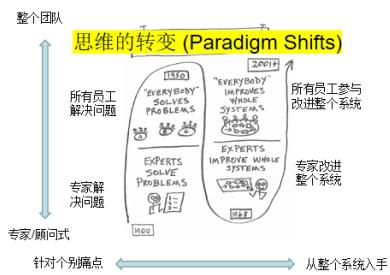
注意：人的心态会不断（按环境）循环变化。

#### 技术创新 Pocket Innovation

软件工程在技术方面变化很大，从我接触的公司，如果一直都没有使用一些新的技术平台，这类公司可以改进的机会极低。反过来，如果公司在过去做过一些内部的改进，比如引进一些自动化技术，推行持续交付、自动化测试等等，就看到这个公司其实已经在技术方面有很多不断提升、完善 的项目在进行中，这个文化就更容易推动所有参与、改进整个系统的可能性。

#### 结束语

过程，改进需要每一个部门都参与，才可以提升公司的最终目标，公司是一个系统，所有员工参与改进整个系统：



(上图参考 M. Weisbord, Productive Workplaces Revisited)

如果没有按照上面说的步骤，形成具体行动计划，很难取得效果。

过程改进不是一两个月、两三个月就出成绩，开始时候，员工可以关于创新、改进有想法，写文章分享，但要这个作为习惯继续下去，是需要有些新的想法和思路，这可以借助公司内部培训，比如有没有过程域相关的参考资料，内部分享会，学习小组，定期有老师的辅助指导，定期体检、诊断。

## 参考 References

1. McGregor, D.: *The Human Side of Enterprise*, 1960.
2. Weisbord, M.R.: *Productive workplaces revisited : dignity, meaning, and community in the 21st century*, 2004.

# Chapter 3

## 克服拖延症

技术总监问：现在我遇到最大的难题就是如何提升下面技术人员的能力，如果他们全都是高手，我就很轻松了，但实际上高手最多只有三分之一，其他都是中低水平。您接触过这么多软件开发团队，有什么好方案？

我：你可以先听听以下故事。

=====

### 目录

Part 1	Beating Procrastination	Part 6	Productivity Hacks
	克服拖延症		产生效率黑洞
Part 2	Becoming Organized	Part 7	Doing the Right Work
	做事更有条理		做对的工作
Part 3	Staying Energized		
	保持活力		
Part 4	Getting Things Finished		
	把事情完成		
Part 5	Automate Your Routine		
	让你的日常工作自动进行		

例如，第一章克服拖延症，这里的内容几乎全部都有帮助：

### 周 / 日目标 (1 Weekly / daily Goals)

我每天都会定计划，早上希望完成哪些功能，下午完成哪些。当然这个计划也会按实际的进展调整。

周 / 日目标是个人时间管理的基本功。每一天第一件事不是回邮件，而是仔细想想今天要完成什么任务，每一周的开始，也应该想我本周希望完成什么任务。不然的话，每天的时间就很容易被琐碎的小事吃掉，一事无成。

背后体现的道理很简单，要把时间花在重要、但非紧急的活动上，效率才会体现出来。

	紧急	非紧急
重要	危机 紧迫的问题 受截止时间驱使的项目	避免、提升生产力 建立关系 新的机会 规划、娱乐
非重要	打扰，电话 邮件，报告 会议 紧迫的事情 受欢迎的活动	琐事，繁忙的工作 邮件 电话 浪费时间 令人愉快的活动

### 限定时间 (Timeboxing)

把每天的任务安排成时间段，每一段不应超过 1.5 小时。一般人可以专心集中的时间段都不会超过 60 分钟，小孩可能更短。如果老师叫你星期五 5 点钟交卷，你不会提前交，都会等到最后十分钟，甚至最后五分钟。所以如果我们把一天的时间切开，分成 1 ~ 1.5 小时时时间段，自然有动力，希望在时间之内完成任务。我们写代码的时候应该也是用同样的原理。例如某些编程活动尝试了多次，但没有进展，有时总会花超过 10 小时。所以每次当我发现某编程工作超过了 2 小时，我就会先做其他事情。

### 分解任务 (Dissolving tasks)

因为都是练习题，所以每一个功能都比较细，不会超过二十行。如果我们平常做开发时，也必须要把一些大、复杂的功能预先拆分成小的功能才有效率。

### 加强自律 (Building Self-Discipline Muscles)

#### 晨礼 (Morning Rituals) 日常运动 (Make an Exercise Routine)

不要以为编码是一个单纯的脑力活，整天坐在电脑前面敲代码就可以。如果人的体力、精力没有配合上也会出问题，好在我每天早上一直坚持 30 ~ 40 分钟的轻量运动，然后晚饭前半个小时到一个小时的骑单车或者慢跑的习惯。中间也不是整天坐着，一段时间会走一走、喝点儿橙汁等，以确保身体不断在动，这样才不会困，保持动力。贝多芬每天都会通过去外面散步来获得一些创作的灵感，然后他会立马把这些写在本子上，用于后面的音乐创作。

我：身体健康，精神状态也同样重要，你每周有锻炼的习惯吗？

小李：没有，每天都太忙了，虽然一直觉得身体不如几年前了，也知道锻炼好，但无法抽出时间。

我：我也很忙，但深知定期运动对身体非常重要，我一直按以下两方法保持个人身体状态：

- 工作时尽量避免长期坐下来，因我主要做培训、咨询、评估，所以可以大部分时间站着或在走动（NEAT#）
- 尽量每天 7 点吃早餐前跑圈，疾跑 1.5 - 2 分钟，休息半分钟，重复这循环 4-6 轮（HIIT#）

### 不会分心的工作场所 (Create a Distraction-Free workplace)

#### 轻策划、迭代、再策划 (Ready , Fire, Aim!)

三十年前，软件开发都是一些大型的项目，整个架构要设计好才动手去写代码。现在反过来，需求变化极大，开发都需要敏捷，轻文档、轻计划。尽快写好代码，做一些功能给客户，从反馈优化下一轮。我这次的几天开发也是用同样原则，没有花时间在一些设计或者文档。想直接把代码写出来，并通过单元测试，节省了很多耗时间的工作。把有限的时间都放在写好代码上。

### 不断清洗 (Churning)

万事开头难。我在开始的半天也是遇到同样问题，不知如何入手，太久没看写代码的书了，很多基本的都不知如何入手。所以我开始的时候不会直接尝试写题目里面的功能，而是重写一些书本的代码，看看跑出来怎么样，然后逐步提升。写一些基本功能，慢慢有了习惯，调整过来了，后面就越来越顺。好比一台旧的水泵，刚开始抽上来的水总是有难喝的铁锈，只要不停止抽水，当污水最终都从系统中抽出后，就能发现底下的净水。

### 要有好的土壤 (Remove your Hidden Roadblocks)

在含盐量高的土壤里种植物是结不出果实的。浇水、平衡在阴凉处和阳光下的时间都抵不过根部吸入的毒素。如果我们没有积极性，就可能是土壤的问题。如果没有足够的积极动力，就不会在长假专注写程序，也不会定期要求自己写分享文章。所以要有明确、很想达到的目标驱动。像一个作曲家，他希望写出很多经典的优秀作品，不满足于现在的状态。觉得自己的灵感或者创造力没有发挥出来，成为可以保留下来的东西。也是这种驱动力让我可以一直努力做这件事。

### 摒弃拖延恶习 (Quit your Procrastination Vices)

长假里，大部分人都会把时间用于看视频或电视剧，而我正好没有这个习惯，也一直没有玩网络游戏的习惯，否则肯定完成不了。

最终我用日程记录 (Timelogging)，把整件事和什么活动、时间花在什么地方都记录下来了。

小李：我看你上面列出的技巧，我大部分都还没做到。

我：不要紧，我六年前刚开始定期写文章时跟你一样，但只要不放弃，一直往既定目标努力，不良习惯都改正过来了。我常常说人的潜力是极大的。舒伯特你

听过吗？

小李：好像是一个很有名的作曲家。

我：是的，但他 31 岁就去世了，你猜他一生一共写了多少首歌和音乐作品。

小李：我记得中学时，老师介绍过他的艺术作品，如《鳟鱼》，但他 31 岁就死了，我猜 100 ~ 200 首歌？

我：他一生写了超过 460 首歌曲（时长 >24 小时）。除了歌曲，他还写了其他作品，如 9 首交响曲（1 首未完成，1 首只有草稿），20 室内乐，120 钢琴曲等，每一类都包括大量经典作品，对后世影响深远。

小李：如果粗算一下，他一生约有 600 作品，算他有 16 年时间作曲，平均每月要完成 3 个作品，真是不得了。

我：虽然他的作品有大有小（从一首歌到 45 分钟的交响曲），他确实生产率极高，而且他最后的 7 年身体一直都不好，所以他那个时候肯定不会像我们现代 996 方式工作。他每天主要是早上用来写作，傍晚便去休息散步。但他会同时做多个创作项目。如果项目没有灵感，就暂时放下来，创作其他作品。他著名的未完成交响曲就是个好例子，只有两个乐章（一般交响曲都是四个乐章）所以他是使用高效技巧的一个成功例子。

每个人都有自己的理想，但如果没有高效率来执行，理想只是天马行空，天方夜谭，不会有任何成就。除了以上这些技巧外，保持整洁也重要。你有没有试过想找某东西，找半天都找不着？

小李：确实经常发生，而且还会遗漏东西。我上次出差便忘记了 iPhone，后面回北京后电话联系当地酒店前台后，我找当地同事去酒店取，然后快递给我，烦死了。

我：有听过 5S（5S 法 #）吗？例如，如果你把东西都放固定地方，就可以避免同类问题再发生。如果你一直在一个很乱的环境工作，会导致心情烦躁，对工作、身体都不好。

（# 详见附件‘5S 法’，‘锻炼之道’多了解 5S, NEAT, HIIT）

小李：我大概懂你的意思了，要提升自我能力先要改变习惯，有了良好习惯（如时间管理），才可能提升。

=====

总监：我大概懂你的意思了，要提升技术人员的能力先要改变他们的习惯，有良好的习惯（如时间管理），才有机会提升。

即时笔记 (The Capture Device) 总监边听边在本子上记下那些重点。高效的人都会有工具帮他记录想到的灵感、想法、项目、待做事项等，不会仅仅靠大脑记忆。你提出一个要求，他会立马写在小本子上，你会觉得他应该会按你要求去处理，但反过来如他只是口头说会处理，你会担心很可能没有下文。但我看有些领导，身边只拿个手机，除非他们的记忆力超人，否则我估计他每天都会忘记不少重要事项。

## 结束语

杭州某高级经理的高见人一定要自律！您说的小技巧确实能起到很大帮助，而且我基本都会使用，但如果不养成习惯，想起来使用下，最终还是改不了拖延症，所以要解决拖延症，一定从根源做起，还是得靠自己，需要培养自己意志力、专注力，坚持好习惯，改掉坏毛病。

我相信人分高低，但并非取决于基因、种族，主要取决于她后天的习惯、自律与努力。要养成良好习惯要从小开始，深受家庭和教育的影响，所以百年树人。

与公司改进一样，改变个人习惯很难，这些技巧可以帮助个人改善。

## 附件

### 5S 法

本来 5S 是用于工业生产，例如日本的生产工厂很注重洁净，东西要放在固定位置，容易找到。其实这对生产线员工起很重要的心理作用，想象如果整个环境都很脏，必然会影响人工做好的动力，如果东西乱放，也容易找不到。5S 也不仅仅是用于生产，比如医院、酒店也采用这种方式管理，比如每件要用的东西必须在固定位置放好，也可以用于个人管理，比如我以前常常在出差去客户时，经常忘记把一些东西，如鼠标，或插头。但后面我固定了每件东西都应放哪里，我走的时候收拾就确保那些东西不会遗漏掉，平常工作有一个洁净的环境，也能降低工作压力，提高工作效率。

### 锻炼之道 The Truth about EXERCISE

想大家都同意和相信：“多运动，便多消耗卡路里，便能帮助减肥，降低体重。”

某国家给市民的健康指南：每周起码做 150 分钟中强度锻炼，或 75 分钟高强度锻炼。

但不是每个人都能每天抽时间做锻炼，有什么更好方法？

不一定只依赖去健身房锻炼，平常工作生活，少坐，多走路，站着工作，开会，甚至小动作等都有帮助。

N.E.A.T. (Non-exercise activity thermogenesis) 小实验；教授使用有电子传感器的底裤，记录记者，咖啡厅女服务员，商务人员三人一周每天正常工作中消耗多少卡路里。

发现

- 女服务员最好 --- 因每天都非常忙碌（尤其是早餐时段）--- 送餐，接单，做咖啡等等。
- 商务人员第二名，虽然有很多时间坐下来，但每天都会走一公里路见客户，而且每周二，五下班后会去健身锻炼。
- 记者最差：每天无论工作或家里，大部分时间都是坐下不动，所以他看起来不胖，但其实体内存有大量脂肪，集中在肝，肾等内脏旁。

这实验告诉我们：如果每天一直坐下，不动，很不好；就算每天下班后晚上都去健身锻炼也帮不了。

后面记者听教授建议，改变习惯，定期站起来走动。如与同事交流与尽量边走路边交流，减少长期坐下来；多爬楼梯，少用电梯；骑单车，不开车等方式。后面，从实验数据分析，发现这些改变帮他增加每天卡路里消耗接近一倍，到 500 水平。

研究发现不断大量健身锻炼不一定对每个人都有效。有 20% 会没有效果，另一端对 15% 的人会非常有效。这跟人的基因密切相关，所以多锻炼不一定都有效。

=====

实验发现，如锻炼能快速提升心跳率到最高，然后休息，反复做 4 - 6 轮，效果不会比大量健身锻炼差。例如，每天做几轮 40 秒的冲刺，把心跳速度快速提升到极限，效果可以比长时间的缓步长跑更好。

HIIT (High Intensity Interval Training) 小实验：教授与助手教记者使用运动单车做 HIIT：用尽全力练 20 秒，休息，再同样做两轮；每周三次。  
记者按教授要求完成了四周 HIIT 锻炼，虽然帮他提高了血液分解糖份的能力，减少糖尿病风险；但提升不了他的最高带氧运动量。教授解释这是因记者的遗传基因是属于没有效果的 20%。

## 参考 References

1. YOUNG, Scott: "The Little book of Productivity" 《超效率手册》

## **Part III**

### **自我管理开发过程**



## 软件开发工程师自我改进过程



# Chapter 4

## 三点估算



## Chapter 5

量化管理从个人开始



# Chapter 6

## 估算软件规模



# Chapter 7

## 估算工作量



## **Part IV**

### **开始过程改进之旅**



团队必须先获得管理层的关注与支持



# Chapter 8

## 获取高层支持

我：对过程改进来说，最重要的成功要素是什么？

客户：最难的是如何得到高层的支持，这不仅仅是嘴巴说说而已，而是要切实地给人、给时间。高层往往不清楚什么是质量改进的重点，但他们对员工的人均收入、利润（比如员工可为公司盈利的时间占比。如果少，就表示这个员工对公司的盈利贡献不够。）等这些财务指标都非常清楚。

我：非常赞同。我们可以利用评估机会来引起高层对质量改进的重视，但往往评估组只说软件开发的各种问题，难以引起他们注意。一般人，尤其是高层，一听到问题都会觉得比较烦，没有动力听下去，更不要说有对应的改进行动了。

客户：你说的挺有道理的。确实我们每年都会有一些评估，发现了不少问题。高层也都会参加评估会，并同意这些问题需要改进。但很多时候过后就没下文了。

我：质量大师裘兰博士 (Dr JURAN) 有丰富的过程改进的经验，他深知要说服高层真心投入、支持改进，就必须用高层的语言打动他们。如果与高层交流，不能把团队关注的事转化成高层关注的，就难以获得立项，做改进。

我：所以在最近的某评估里，我会引导公司内部的质量经理，跟公司高层汇报时，着重汇报能为公司节省成本的初步方案。你有兴趣听听这案例吗？

客户：非常有兴趣。

== 案例 == 我问公司内部评估组，高层有哪些关注点？他们就列出来，如人均收入、人均利润等商务指标。

然后我再问影响这些指标的因素很多，有些是软件开发团队无法控制的，例如销售人员与客户的关系，所以我们必须从这些高层关注的指标细分到一些团队实际可以影响到的指标。有哪些呢？

质量经理：项目进度的偏差，遗漏给客户的缺陷数。

我：这些我们都叫性能目标。这些指标是如何影响高层目标的？

质量经理：比如项目延误，成本就会超支。

我：是的。如果我们过程改进希望改善质量、减少缺陷，你们觉得会对成本有什么影响？

质量经理：改进要投入工作量，肯定会增加成本。但长远应可降低成本。

我：很多高层都不熟悉质量管理，所以很少关注和监控团队的缺陷。他们通常还是会觉得质量改进是好事，但要花成本。我做到客户满意的水平就可以了，不

要追求十全十美，也正是这个误解，才导致他们认为大部分缺陷在系统测试或验收阶段才发现是正常、是常态。如果是开发软件产品的公司会好一些。在软件工程领域，因为缺陷发现越靠后，返工工作量是前期发现同样缺陷的二三十倍。所以如果我们能把返工缺陷预先发现并处理，必然会大量降低返工工作量，与相关成本。

{| class="wikitable" |- | 理论上也可以用缺陷排除率来估算改进后的缺陷分布（汇总第5部分细讲），但与高层沟通越直接越简单越好，所以只需要简单估计改进后的数量，大家觉得合理便可。|}

## 第一版方案书

质量经理做了第一版的方案书。

项目A: 计划60人月 150w					
活动	单个缺陷返工工时/h	原缺陷数	原返工工时/h	改善后缺陷数	改善后返工工时/h
需求评审	2	10	20	15	30
设计评审	2	5	10	8	16
UT阶段	2	45	90	60	120
SIT阶段	2.5	240	600	180	450
UAT阶段	3	100	300	70	210
总计			1020		826
成本占比			10.20%		8.26%

改善效果：成本下降1.94%，约2.91万

问题：

影响：

造成损失，成本增加8.3%，约12.5w

范围变更，管理缺失

项目延期，成本增加

60人月 → 65人月

150w → 162.5w

我：挺好的。你在投资方面是保守型吗？

质量经理：不是啊，我还会定期买股票，因为单是靠银行定期，利息太低了。

我：但是看你在质量改进方面的目标很保守。比如你看，说如果做了改进以后，后面的缺陷可以降低百分之十到百分之二十，然后最后算出来节省不到五个点。你估计高层会买单，投资你这个项目吗？

质量经理：确实有道理。

我：你这里有几个地方没做对，例如从我们过去的经验，因为很多团队在前面几乎没做好单元测试或代码扫描，反之，系统测试或者验收测试的缺陷几乎可以减少一半。低的目标不是好目标。还有，你评估在后面阶段返工的工作量也太低了，只是比前面发现的高一点点。你尝试依据这两个思路再调一下吧。

质量经理：你有所不知，我们虽然有些行业参考数据，但没有实际项目数据支撑，是否需要先找些实际数据才可以做方案，不然好像说不过去。

我：在初步方案阶段，其实不需要实际数据的支撑。你可以想象，目的是要让管理者有依据做决定，你可以利用实际数据把节省算得更准确，但对高层决策没有太大帮助。所以在初始阶段，合理的估算就足够了，不需要花精力在没有价值的事情上。

## 第二版与中层确认，然后向高层汇报

做了第二版后，大家觉得这确实比之前更有说服力了，质量经理再跟中层确认。

项目A: 规划120人月, 总成本300万						问题	需求变更 评审不充分（例如：设计 评审） 单元测试（自动化脚本） 开发人员技能不足 版本管理问题
里程碑	单个缺陷 返工工时 /h	原缺陷数	原返工 工时/h	改善后 缺陷数	改善后 返工工时 /h		
需求评审	1	10	10	15	15		
设计评审	1	5	5	8	8		
UT阶段	1	45	45	60	60		
SIT阶段	20	240	4800	120	2400		
UAT阶段	30	100	3000	50	1500		
总计			7860		3983		
成本占比			39.30%		19.92%		
改善效果: 成本下降19.39%, 约58万						51%	

质量经理跟高层汇报，并得到高层的认可后，高兴地说：以前都以为过程改进应先自己默默耕耘，先在试点取得效果，再跟高层要资源后再推广。现在看，还是应该一开始便提具体改进方案并立项，才有机会成功。

我：管理层是过程改进最重要的干系人，必须一开始便得到他们支持。你后面有什么计划？

质量经理：既然他们都同意，我就开始找项目试点，对吗？

我：虽然试点很重要，但不是第一步。你们一直有过程改进小组吗？质量经理：懂你的意思，但实际没有。团队有关质量问题便会直接找我。我：应让团队自己依据迭代数据，分析根因，下一轮做改进。但有些需要跨功能或组织级改进，如完善复用框架，或完善规范/指南与相关检查单等，便需要依靠某过程改进小组来推动了。如果你们没有，便应尽快举办2天工作坊互动培训，邀请所有过程的利益相关者参加，一起讨论未来改进重点并制定具体短期长期目标与计划。

质量经理：你说的有道理，请发我相关案例，打铁趁热，我尽快跟高层说。

:: (关于2天工作坊的实例与照片，会在13章细讲。)

### 反馈

某 CTO：方向很好，但高层关注的不只是销售额这些商业指标，还有交付效率和质量的指标。

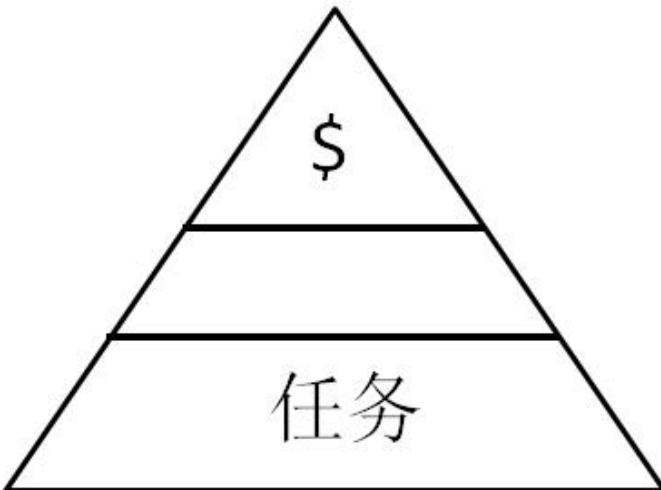
公司老板针于研发除了关注成本外，也关注以下内容：

- 人员：人均薪资、人均产能、人员参项率（有收入的项目）、人员闲置率
- 进度：是否能正常交付
- 质量：缺陷率、严重缺陷数

举例说明：比如提升效率这样的指标，多数公司都用人均产值，人均交付金额这些指标。但大多数公司没有想过创新，如果公司业务很稳定，那么针对重复性工作是否能形成公共组件或低代码平台，来大幅度提升生产率。质量方面，如果客户能参与使用，例如产品定义，需求的质量会大幅度提升（不仅依赖评审，测试）。对于这个逻辑，凡是做过程改进的人都能理解，但为什么客户不愿意深度参与，是因为没有让客户可以高效参与的工具。

例如 90% 以上的公司都是靠需求文档，RP 原型的方式沟通需求，效率不可能高。但如果低代码平台能快速形成需求和客户可体验的需求，效率和质量都会大幅度提升。

我：是的，大量缺陷在后期才暴露只是最常见的质量问题。反映团队的质量水平，而让客户多参与需求，低代码平台是一些改进方案。但想要过程改进成功，首先必须得到高层的支持，但高层往往只关注日常任务，不一定关注质量，所以必须靠中层作为高层与下面团队的桥梁，把质量改善简化成高层可以理解的方案，让他决策。



很多高层只关注项目是否按时交付和财务指标，虽然大家都知道单靠财务指标是不全面，但通常只有财务数据。所以在上面质量经理提交改进方案这案例里，我强调质量改进方案必须重点分析能节省多少人力成本，不仅仅是减少最后遗漏到客户的缺陷率。但如果面对像你这种通情达理的高层，就可以更全面，从质量、创新等多维度估计改进能如何为公司带来价值。

## 总结

从以上简单案例，得到的经验教训：

- # 有效的质量改进必须从获取高层支持开始。
- # 中层要从高层关注点（ $\bar{Y}$ ）提改进方案。
- # 在做改进方案时，只需要估算，暂时不需要实际项目统计数据，以此可以节省资源和时间。
- # 如果确实有高层真正支持的话，就必须正式立项，再配上具体的团队、分工和详细计划，否则整个改进计划还是一纸空文。

1. 立项后，可以用 2 天工作坊培训启动。



## Chapter 9

### 寻找改进机会



## **Part V**

### **如何做好迭代回顾**



怎样避免同类问题的重复发生？因为有很多问题背后的原因是整个系统的不足，而不仅仅是个人不关注/不努力，做好根因分析可以帮助我们找出背后的根因。

怎样做好根因分析？二八原则、五个“为什么”（5Why）等，大家可能都听过。在网上也可能找到相关的理论知识。但很多人还是有误解，没有弄清楚应该如何利用这些技巧来找出根因。

针对软件开发的过程改进，也可以使用根因分析，帮助软件开发团队避免问题的重复发生，以提升团队的质量和效率。

迭代回顾（Retrospective）可以让团队每走一小步（一个冲刺），便立马回顾过程中有哪些不足、分析根因，以便在下一小步中得到改善。

要做好迭代回顾，除了要了解根因分析的技巧，还要了解回顾背后的原则，配合团队互动练习，再配合工具做好数据分析，才能使团队有实际的提升，让老板满意。

这部分会讨论：

1. 根因分析的主要元素
2. 软件开发团队应针对哪方面入手，才能最容易取得效果
3. 迭代回顾前应如何做好准备
4. 内部教练引导团队做回顾时应注意什么
5. 如何可延续，让团队（甚至组织级）可以建立标杆



# Chapter 10

## 根因分析

### 分析事故根因：上集

北京的某软件开发公司，专门为电信供应商做定制软件开发与运营，比如用短信做些推广活动等。公司希望做过程改进。

我：过程改进的主要目的是帮助企业更好地达成业务目标。你们应该是最清楚自己有哪些不足的，请问你觉得哪些方面有改进空间？

总经理：我不太熟悉技术细节，只能从客户视角来看。例如，去年因为软件的一个错误导致客户损失了几十万元，我立即在公司内部开会，希望找出根因，最后发现是因为某个开发人员粗心大意，在编码时把两行数字搞错了，在测试人员针对这项功能进行测试时，虽然收到短信了，但并没有仔细看短信的内容，也就没有发现这个缺陷，测试通过了。这起事故不仅招致了公司的经济损失，更影响了客户对我司的信任度，也影响公司在行业内的口碑。

我：公司采取了哪些纠正措施？

总经理：之后我们加强了对代码的评审工作，要求这类代码都需要经过主管审查，还增加了相应的惩罚措施，如果再次发生同类事件，项目经理和部门负责人都要承担连带责任。

我：采取这个措施之后，效果如何？

总经理：很好，大家都注意了，到现在都还没有再发生同类的问题。

====

你认为以上的纠正措施能避免同类问题的再次发生吗？

你可能赞同以上事故的主因是人员疏忽，但很多事故的主因其实不仅仅是人员疏忽。

## 香港南丫海难

2012 年 10 月 1 日晚上 8 时，任职港九小轮的黎细明驾驶海泰号（注）由中环出发前往南丫岛榕树湾。15 分钟后，任职港灯的周志伟驾驶南丫四号从榕树湾出发，接载港灯员工及亲友到维港观赏国庆烟花。两船于 8 时 20 分相撞，南丫四号船尾迅速沉没，造成 39 人遇难。  
(注：海泰号是大型喷射双体船，速度与载客量都远比南丫四号高。)

南丫四号船尾沉没，消防处安排潜水员紧急救援：



黎细明 (左，56 岁)，小二文化程度，1982 年任职小轮公司，2008 年成为船长。





周志伟 (58岁), 小六文化程度, 已婚, 育有两女, 1974 成为海员, 1993 年开始担任船长。

**2015年, 开庭审判两船长:**

周: 碰撞发生前, 南丫四号曾右 3 度避撞, 但海泰号违规左转 3 度, 且海泰号船速较快及操控性较高, 故南丫四号的避撞措施均被海泰号的相反举动抵消。

黎: 造成大量乘客死亡的原因是南丫四号船身有问题, 包括部分船舱并非水密舱等, 而且海上的雾灯及其他背景灯光也影响了我的视线, 难以观察到南丫四号。

专家: 海泰号没有根据“国际海上避碰规则”右转, 反而左转 3 度, 属“极度危险”, 南丫四号虽曾按规则向右转, 但“幅度太小、转得太迟”。

- 陪审团以 7 比 2 裁定黎细明误杀罪成立, 法官判入狱 8 年
- 陪审团以 8 比 1 裁定周志伟误杀罪不成立, 但危害他人在海上安全罪以 7 比 2 裁定罪成, 法官判入狱 6 个月

你赞同两位船长的失职是本次海难的根因吗?

政府经过 3 年多的事件调查, 最终做出调查报告。报告显示, 在海难中沉没的南丫四号, 从设计到验收, 每个环节都存在纰漏, 每个环节都有出错, 例如包括:

- 没有安装水密门 (导致南丫四号在碰撞另一艘船之后, 迅速沉没)
- 座位的螺丝松动 (导致撞船后, 因船尾沉没水中, 所有座位都松脱, 跌到船尾, 影响乘客逃生)
- 没有儿童救生衣等

因此导致南丫四号在碰撞海泰号之后, 迅速沉没, 并且导致罹难人数众多, 政府展开内部调查, 怀疑海事处有十七名职员应负责任。

除了南丫四号的问题, 你是否觉得小轮公司也应负责任, 例如:

- 对船长的培训/监督是否足够
- 是否有定期检查船上救生设备

你可能要反驳，像“南丫四号”这类小型船，预算少，所以关注度、监控度都低，如果是大型项目应该不会出现这类纰漏事故，请看看美国 NASA 的太空穿梭机 (Space Shuttle) 计划因发生了两次致命的事故，最终被叫停的案例。

#### 太空穿梭机灾难 (Space Shuttle disasters)

1986: 挑战者号 (Challenger)，挑战者号航天飞机升空后，于发射后的第 73 秒爆炸，机上 7 名宇航员全部罹难。(相关视频可以在网上搜到)

2003: 哥伦比亚号 (Columbia) 航天飞机在返回地球时，因为左翼的隔热保护胶在十六天前升空时被固体火箭助推器外层脱落的乳胶击破损坏，导致航天飞机返航时进入大气层的第 1000 秒钟，在 35,000 米高空因过热解体，7 名宇航员全部罹难。

Sally RIDE 在 2003 哥伦比亚号灾难事故调查回顾说：我很诧异这两次事故的原因非常相似（她也是挑战者号事故调查组员），86 年引起挑战者灾难的陋习又再出现：为了赶上计划升空日期，而忽略了之前性能报告的发现，没有注意前线技术人员提出的安全隐患，也缺乏安全保障措施。其实从哥伦比亚号在 1981 年首次升空开始，每次都有出现升空时燃料缸外的保护乳胶掉落的事故，但管理层一直都没有关注，工程师因为赶进度，每次升空前，都没有时间预先处理好。



如果管理层、工程师只关注项目的成本与进度，忽略了质量与安全，便会导致灾难。

其它很多相似的航空或海上灾难，虽然当事人（如船长）的失误引发了事故，但这并非根因，背后系统的不足才是主因。

上面总经理的纠正措施能避免同类问题再次出现吗？

不一定。因为已经导致公司极大的损失，而且引发了高管的关注，开始的时候大家会注意，但并不能长久，因为这仅仅依赖于个人的习惯，很可能在几个月后还会再次发生。

## 根因分析的主要元素

如果想挖掘系统的不足，便必须找出根因，根因分析主要包含那些元素？可参考 CMMI 模型的根因分析 (Causal Analysis & Resolution CAR):

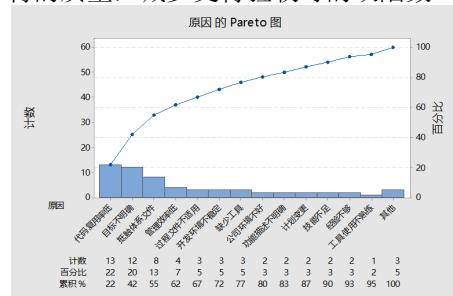
V1.3	V2.0
CAR sp 1.1	选择要分析的结果 Select outcomes for analysis
CAR sp 1.2	分析原因 Determine Root cause
CAR sp 2.1	对应措施（改进建议） Address the causes (Improvement proposal)
CAR sp 2.2	评价改进效果 Evaluate the Impact
CAR sp 2.3	记录原因分析数据 (A3报告) Record Casual Analysis Data
	CAR 2.1
	CAR 3.1
	CAR 4.1
	CAR 3.2
	CAR 3.3
	CAR 4.2
	CAR 3.4

1. 利用二八原则识别引起大部分问题的最主要因素
2. 分析背后的主要原因
3. 对应改进措施
4. 判断改进效果
5. 总结成根因分析报告

可参考附录 A: ”绅士俱乐部过程改进”了解 QC 圈如何按意识根因分析元素完成为期七个月的过程改进。本章附件也有二八原则与帕累托图的介绍。但不要误以为只要使用各种根因分析方法，如帕累托图、鱼骨图等，就能做好根因分析。

## 根因分析误解案例

某公司过程改进组（共 4 人）分析以往一年项目，做根因分析，希望改进交付的质量，减少交付验收时的缺陷数。



我：请问这帕累托图的依据？

组长：针对客户缺陷密度较高的现状，我们做了敏感度分析。发现需求引入的缺陷数跟它相关性最高，我们接下来就用鱼骨图分析，四个人一起头脑风暴，识别出以下十几项主要的原因种类，然后我们就按每人三票，各自单独投票得出这个排序，然后我们就依据二八原则选择了头三项原因是主要的对象，然后我们根据这三项原因发现，这些项目是导致需求缺陷的原因，包括流程图画不好等。

我听完以上根因分析故事，便在投影仪投出以下某机场某年导致航班延误的原因统计表，问项目经理：“假如你是顾问团队，要为机场管理层高层分析导致航班延误的主因并提出改进方案你会怎么利用二八原则画帕累托图？”

### 香港启德国际机场 1996 年航班延误统计：

Cause 原因	Count 次数
Late baggage to aircraft. 运输行李到飞机有延误	340
Bad weather. 烈性的天气	240
Too few gate agents. 登机口工作人员太少	66
Late cabin cleaners. 清洁工打扫机舱晚到	21
Poor announcement of departures. 出发通知混乱	12
Gate agents not motivated. 登机口工作人员没有动力，影响工作	103
Late aircraft arrival to gate. 飞机晚到	68
Late or unavailable cabin crews. 空乘（飞机服务）人员迟到或因故不能上岗	21
Passengers bypass check-in counter. 旅客绕过出登机牌柜台	62
Late passenger cutoff too close to departure time. 最晚乘客登机时间太接近起飞	44
Late food services. 飞机餐饮服务晚到	37
Gate agents undertrained. 登机口地勤人员训练不足	116
Desire to "protect" late passengers. 希望“保护”迟到的乘客	57
Late issuance of boarding passes. 打印登机牌发生延误	33
Late weight and balance sheet. 飞机重量平衡表（为了确保飞机重量平衡，保证飞机安全）迟交	40
Late or unavailable cockpit crews. 驾驶舱人员迟到或因故不能到岗	19
Confused seat selection. 座位安排混乱	66
Late fuel. 燃料延误	35
Gate agents arrive late at gate. 登机口工作人员晚到登机口	19
Late pushback tug. 推车延误	75
Checking oversized luggage. 检查超大行李	20
Desire to help company's income. 希望帮助航空公司增加收入	37
Air traffic control delays. 航空交通管制导致延误	166
Poor gate location. 登机口的位置不好	130
Mechanical failures. 机械故障	26
Delayed check-in procedure. 登机牌柜台手续 / 过程发生延误	31
Gate occupied by a late departing plane. 停机位被前面晚点起飞的飞机占据了	22

然后我解释：

我们不应直接把几十条原因，按每条原因对问题的影响度，从最高排到最低做帕累托图，而应先把原因进行分类。用上面机场延误为例，比如，哪些是不可抗力的天然原因（如天气原因），哪些是跟闸口管理相关的问题，那些是跟机场打印登机牌相关等，例如，如果识别出是跟闸口管理类的问题最多，后面便针对这进一步分析根因。分组才有针对性、有意义。

你们的分类，帕累托图依据的“数据”，都只是用数字形式表达你们头脑风暴的主观判断，所以你用这种帕累托图其实跟直接用头脑风暴做出做出的结论没什么差异，只是你们用了投票数据使结果看起来“量化”了，其实是没有客观数据支撑。

例如，你们的分类有问题：例如，只有“功能描述不明确”，难道“非功能需求描述不明确”（如性能、安全性等）就不需要考虑吗？

你们现在的分类有不少是重复的，同样一个问题有可能归属于两个、三个分类。

你们度量操作定义有问题，例如，你是如何根据历史项目的数据去判断哪些原因归属于“功能描述不明确”？

如果想做好，应该预先按你们想要解决的问题，识别出相关的度量项，然后针对这些度量项，例如，缺陷，定义如何分类（详见附件“二八原则与数据分类”）并收集数据后，做分析。有了客观的数据分析，才是你们根因分析的第一步。

====

我：你们现在的找出的其实并非根因，只是浮在水面上的现象，你们有听过 5W 吗？

组长：有，What When ..

我：不，你说的是 5W+1H，5W 是五个为什么 (why)，所以你们应一直追问，才能更好地识别出背后的主因，并针对主因采取纠正措施，才能避免同类问题

重复发生。

我继续向他们讲述丰田汽车大野耐一先生的 5W 故事。(详见附件“大野耐一先生 5Why 例子”)

除了通过问“为什么”之外，模型(例如 CMMI, XP 极限编程等)也可以帮你们更全面地找出根因。很多原因会导致项目延误。“功能描述不明确”可能只是其中一个原因，例如：

- 利用检查单做好需求评审
- 与干系人(客户)确认需求

如果能做好这些，即可帮助团队确保需求质量，减少项目延误，以上两条都是 CMMI 里的最佳实践，所以模型可以帮助团队更全面地看问题。

### 怎样才能做好根因分析

从前面各例子看到，有些团队没有根因分析的意识：不理解核心思路，只是表面“做”了根因分析的步骤，其实未找到根因。所以，要做好便必须利用培训，提高大家的根因意识。大家从以下日本回国技术总监的故事可以更好理解什么是“根因意识”。

### 技术总监经验之谈

总监之前一直在日本带领开发团队，十年前回大陆。最近为他们做过程差距分析，总监开车送我去机场时分享他的日本经历。

你说我们团队没有找出真正根因，我非常赞同。日本人在根因分析方面做得特别好。我之前在日本工作，带领小组做开发，当时我们团队共 5 人，有 2 位来自大陆，其他是本地人，有一次，因为我们开发计算公积金出错，QA 一直追问问题的原因。当时我还没有找出根因的概念，不明白 QA 的意图，以为只是问责，希望找个人背锅。其实他们确实是希望找出问题的源头，避免同类问题再发生。当时主管追问原因，要求我发问题报告。我的报告只说了一些开发问题，人员经验不足等。主管一直追问为什么。“如果说培训不足，你有什么相关的培训计划来避免?”我回答不上来。最后发现，原来是日本计算方法和大陆不同，他们不四舍五入，导致我们两位大陆开发人员的计算便和本地的不同。很多大陆工程师没有根因分析概念，认为质量依靠个人保证，“我负责，有问题我承担”。日本人不是这个思路，他们希望挖掘到问题的根本原因并避免。后面我回国发展，开始时兼职做咨询工作，帮一位老朋友看看某软件开发公司团队的问题，发现很多大陆团队对质量方面的要求远远不如日本，很不习惯。

### 日本产品质量

98年，我在香港富士通工作，试图在香港市场推富士通的服务器，富士通服务器跟美国太阳（SUN）匹配（compatible），但价位反比美国太阳服务器高，一直想不出什么原因。后来我去东京出差，发现原来日本公司，例如富士通，对质量特别有要求，产品测试设计等都要通过很多关卡，才可以出厂。有些富士通在澳大利亚的工程师就不明白为什么总公司要求这么苛刻，需要不断测试，觉得浪费资源。他们不清楚原来日本本土客户对质量要求特别高，无论是消费品或电子产品，如果达不到高水平质量要求，基本上卖不出去。富士通公司电子产品以满足本土市场为主，所以必须有很高的质量要求。

客户（甲方）是否对质量有高要求也是促进产品质量的重要因素。

所以如果公司高层没有产品质量的要求，单靠团队学习根因分析技巧不会有改进效果。

### 结束语

根因分析包括下图各主要元素：



从上面案例，大家看到如能用好根因分析，应可帮助团队避免同类问题重复发生，帮团队提升。

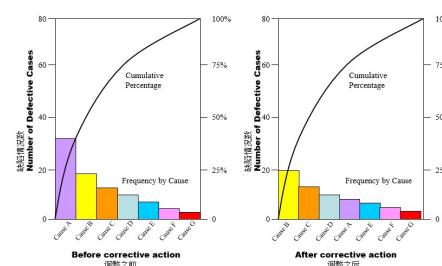
但要真正做好根因分析不能单靠学理论与方法，必须利用实际数据让团队成员动脑筋和讨论，才能真正用好根因分析，取得效果。

下一章，我们探索针对软件开发如何利用根因分析做好过程改进。

### 附件

#### 二八原则与数据分类

帕里托（Pareto），16世纪意大利威尼斯人，他发现虽然威尼斯很富有，但财富并非平均分布，80% 财富在 20% 手里，他也发现很多其他分布都非平均。



因为过程改进需要公司额外的资源投入，必须有针对性地找出最容易取得改进效果的原因，才可能拥有最大的成功机会。所以，应该利用二八原则去识别影响最大的几个因素。以上图为例，原本 A 类问题出现最多，针对这类问题做过程改进之后，A 类问题就少了很多，下一轮便应针对最多的 B 类问题做改进。

有些人以为，只需要对某个维度做分析即可，并没有从多个维度进行分析

例如，某纸制产品工厂会计数据显示，八成的产品问题相关成本都归属于 5 类，例如：质量不合格、赔偿、售后现场服务等（共有 20 类）。针对 5 类中最大的一类质量不合格，发现里面八成的成本都由于六个产品引起（共有 50 个产品）针对这六个产品把成本按缺陷种类细分，发现 B 产品断裂缺陷的成本最高，我们就应该针对这一问题研究如何改善。

产品线	缺陷类型造成的损失			
	断裂	颜色不对	厚度不均	切割问题
B	¥16,000	...	...	...
C	...	...	...	...
A	...	...	...	...
E	...	...	...	...
D	...	...	...	...

### 大野耐一先生“5 Why”实例

有一次，大野耐一先生见到生产线上的机器总是停转，虽然修过多次但仍不见好转，便上前询问现场的工作人员。

(1-Why) 问：“为什么机器停了？”答：“因为超过了负荷，保险丝就断了。”

(2-Why) 问：“为什么超负荷呢？”答：“因为轴承的润滑不够。”

(3-Why) 问：“为什么润滑不够？”答：“因为润滑泵吸不上油来。”

(4-Why) 问：“为什么吸不上油来？”答：“因为油泵轴磨损、松动了。”

(5-Why) 问：“为什么磨损了呢？”答：“因为机器打磨金属零件，空气混进了铁屑等杂质，并掉进机器油缸里。”

经过连续五次不停地问“为什么”，找到问题的真正原因（润滑油里面混进了杂质）和真正的解决方案（安装过滤器）。由现象推其本质，因此找到永久性解决问题的方案，这就是 5 Why。



# Chapter 11

## 如何降低软件开发质量成本

软件开发不同于工业生产，因都是人的行为，依赖团队成员自己收集。（工业生产依赖自动化机器，容易得到数据。）

### 数据收集

“如何才能收集到开发质量相关数据，来分析根因，制定纠正改进措施？因为没有度量，便无法谈改进。”这是很多研发经理想解决的问题。

收集软件开发数据有各种困难，如果没有收集到可信的数据，便无从分析与改进。

是否可以依赖加强组织级度量与监控？我们先看看一家过万人，主要提供金融软件产品，公司遇到的困难，公司一直都很强调量化管理，收集各种项目的系数、度量并分析。

### 自动化统计分析

质量部经理：我们每次都跑全量，公司引入低码平台，更多的是在需求设计阶段做好质量保证，所以我们很注重量化质量管理，能否通过自动化来统计分析，如何通过量化或工具方式实现自动评审。

我：为什么要自动化？

经理：从去年开始我们搞度量分析，发现员工就会造数据，结果导致失真，度量哪里就造哪里，所以还是想通过工具代替人工方式，除了能提升效率，也能帮助判断数据是否合理。

现在我们的主管很反感度量，一度量就有人造数据。

我：度量本来是件好事。经理：是的，就是大家知道算法原理就开始造假。因为我们搞了红黑榜，但很多人头脑都很聪明，会想办法，但用于不合适的地方。

我们有很多数据统计分析，比如看测试用例与需求的比例。其实客户发现的缺陷比例也降低了，但因为我们这行对质量特别注重，产品经过多年的逐步演化，

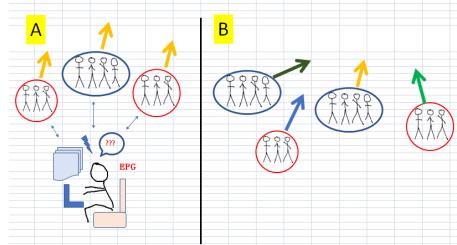
过程很复杂，导致软件缺陷的修复很耗时，客户不太满意。而且公司要求交付的频率要比以前高了很多，我们团队做这些分析都忙不过来，所以需要员工设自动化工具等加快速度才可以。

让我给你看看我们大数据分析。

我：等等，但我们首要解决如何能收集到真正的数据，不然数据分析没有意义。

经理：好的，有什么建议？

我：还记得我们上次交流，要让团队自主，不能单靠标准过程并用指标监控执行情况。



我：请问你们是采用左面还是右面的方法做过程改进？

经理：好像我们现在度量分析是采用你图里左面的方法。

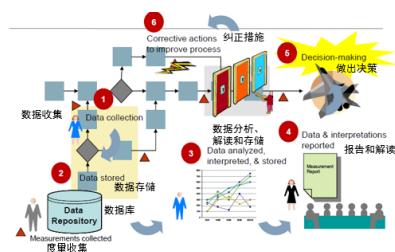
我：是的。总体分析还有另一不足：各个项目特性不一样，你们现在几十个项目总体趋势分析，很可能找不对根因，因每个项目的问题（根本原因）很可能不同。比如，同样是一个测试用例比例数，你的范围就很宽，从最低的 0.3 到最高的超过 200。但你们取平均值 5.1 来做分析。

收集数据也是问题，因为收集数据是挺花精力的工作。

经理：完全同意。

我：但正因为不同项目有各种特性，要对收集到的数据做分析也很耗时。这些辛辛苦苦做出来的分析报告其实不仅仅是给高层（或者项目经理），使每一个团队成员都看到才有意义。（度量分析要反馈回数据提供者，他们才有动力继续收集数据）要把那些分析好的报告再跟每一个团队成员解释要花多大精力？

度量的主要目的是从数据分析找出根因做改进而不是仅仅为了监控



如果把数据分析下放到团队自己搞，便灵活多了；也正因为他们有参与收集和分析讨论，你们也可以节省很多沟通的成本。

所以你们领导应该定位自己是内部老师，辅导团队怎么做好数据分析，效果会更好。

经理：团队自己讨论便可以得到改进吗？不需要我们领导？

我：如果团队有能力，就应该放手让他们自己收集数据、利用数据进行分析并做出改进。你们应把自己重新定位为团队的教练，去辅导他们如何收集与分析数据。千万不要以为这项工作会比以前自己做分析更轻松，其实你们需要更熟悉整个过程，才能真正辅导好团队。但因为你们之前积累了经验，所以辅导团队应该不会太难。更大的挑战反而是要让管理层了解并赞同，敏捷开发需要团队自主的思路。

## 数据收集频率

几个月后，质量经理问：请问老师有没有写过关于客户满意度与产品研发/质量/过程改进之间的关系的文章？我想从研发改进或质量改进的角度，正向推导出如何帮助提升客户满意度的方法。

（公司一直很注重每年的客户满意度调查，高层也会参考满意度调查结果，作为 KPI 系数之一，确定绩效，并决策公司资源的投放等。）

我：我没有写过你说的这类分析文章，但有写过客户的案例分享：某香港公司，他们在广州有开发中心专门为香港的客户做软件维护工作，他们的项目采用 SCRUM 敏捷开发，每两周一个冲刺，他们每次做完迭代后，都会要求客户填写满意度调查表，然后做数据分析。（这分享文章的部分内容，详见附录 B：“分析迭代客户满意度调查数据”）

经理：我刚才看了你发的案例分享，挺好的，但我想要的不是通过分析每一次迭代的客户满意度，让团队做过程改进，我想更宏观一些。每年，我们公司有独立部门做客户满意度调查，已经持续了十年，高层会根据调查的结果判断是否继续向事业部投放资源，也会影响到员工绩效等等，所以其他部门都很关注。但后面越来越发现那些数据的作用不大，因为各事业部都清楚，客户满意度调查的分数很重要，高层也关注，所以都会在调查之前拜访客户，做好准备，确保结果不会太差。

我：你们做了这么多年，应该最清楚自己的问题所在，你觉得现在的做法有什么不足？

经理：我们的客户满意度调查是每年随机抽样一些客户来做，因为是抽样，又是在每年年底才做，有些时候可能项目在上半年的四五月已经结束了。到年底时，客户对很多几个月前的情况都忘记了。

我：是的，为什么不在项目发布后立马就做？

经理：这项工作是由独立部门去做，因工作量比较大，每年都是独立策划安排的。其实我们也有在项目发布后就做的调查，不过时间跨度没有年度这么宽。只是一些简单的高中低打分，由一线人员直接去做。

我：为什么不能安排你们的客户满意度调查也是在项目的结束时间去开展，这样不就更及时吗？

经理：主要是他们的人力有限，资源有限。

我：这个不太说得通，我没有说你们每次发布后都要做调查，这会增加工作量，但你们还是可以随机抽样调查，这样及时性就好多了，不会等到几个月后才调

查。你们为什么只在每年年底做调查并更新标杆？

经理：还有另一个原因，高管很注意客户满意度调查的结果，以此来判断部门的绩效和对事业部的资源投放，所以调查需要与公司年底的预算同步。

我：标杆（基线）的更新不应每年只更新一次，而是实时变化的，如果有显著的变化就更新。还有一个问题，你刚才说因为客户满意度调查的结果会被公司高管用来判断部门的绩效和对事业部的资源投放，所以业务部门肯定会非常关注，并想尽办法得到好的结果，比如预先拜访等等。其实这个不但污染了数据本身的准确性，也影响了数据的可信度，让数据变得不客观。

经理：好的，我会根据你的意见，在明天讨论如何改革客户满意度调查时，提出建议。

## 哪里最耗工作量

软件开发特点，超过 95% 的成本都是人力成本。按二八原则，应先识别最耗工作量的地方？

首先利用二八原则识别出哪类问题的影响最大。在软件开发中，如想提升团队生产率（降低工作量），应先探索哪类工作最耗费工时。

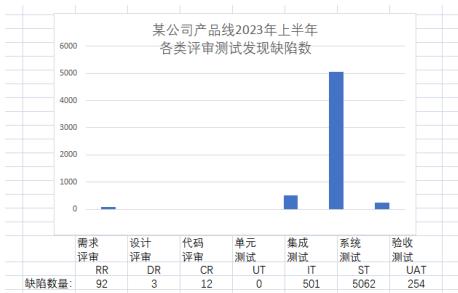
请你把过去一年的软件开发项目，按不同工种占项目总工作量的比例，从最多到最少排个序？

1. 编码与代码设计
2. 交付后的所有工作，包括维护、更新与缺陷修正
3. 交付前的评审，静态扫描，测试与缺陷修正
4. 项目管理与监控

可以参考本章附件中的“开发项目工作量（成本）分布”，看你的选择与典型分布相差多远。

软件开发项目最大的工作量通常是花在找出与修正缺陷上（开发里的‘测试与评审’，详见本章附件），根据软件开发度量专家卡铂斯·琼斯（Capers JONES）先生在 2012 年的研究，对于超过 10,000 功能点、计划使用 25 年的大型系统，有接近一半的工作量是与找出并修正缺陷相关。

很多项目中的缺陷大部分还是到后期才发现，这导致了大量返工。如果能提前发现并解决这些缺陷，就可以大大降低研发成本（不仅仅是提升产品质量）：  
::（绝大部分公司都类似：发现缺陷最多的是在系统测试阶段，验收测试阶段其次）



但很多开发人员误以为编码是项目主要工作，忽视了大量因质量问题引起返工，所以只有当管理层了解了尽早发现并排除缺陷是很好的改进方向，并引起重视，公司才有机会改进。

有些偏业务的领导可能会质疑，客户不一定关注缺陷密度，只要达到可接受的水平就可以了。

我解释“在软件开发中，减少后期测试才发现的缺陷，不仅仅能提升产品质量，更重要的是能降低研发的总工作量，所以最终能帮助公司省钱。”（详见附件“公司高层不一定关注质量”）

## 分析缺陷排除率降低质量成本

质量大师裘兰博士 (Dr JURAN) 强调，过程改进应从认同必须改善质量 (Proof of the Need) 开始。前面第 8 章的案例说明了如后期暴露的缺陷能预先在之前评审或单元测试发现便能大量减少项目成本，主要是质量成本（详见附件）。

但应如何开始？

“应先考虑如何能收集到修复缺陷相关工时数据？因没有度量，便无法谈改进。一般团队（如果用系统管理缺陷）只有缺陷统计数据，缺少缺陷返工工作量，测试工作量数据。”

没有缺陷返工工作量数据，开发团队便不会觉得需要前面预先发现缺陷并解决，会以为应先让测试人员找出问题，开发人员才修正。

下章会介绍如何分析迭代缺陷数据，计算缺陷排除率，并举例说明，若提高前面评审、自测缺陷排除率能降低多少返工工作量。

## 结束语

可以让团队迭代回顾时收集并分析数据，解决公司级统一收集数据的困难。除了收集开发数据，也需要收集客户反馈数据，才全面，但也应每轮交付收集，才能及时分析数据。缺陷返工占软件开发工作量最多，缺陷越后发现，返工量越大。可以分析缺陷排除率，加强前面的代码扫描与评审，单元测试等，尽早发现并解决缺陷，不仅仅提供产品质量，也降低质量成本，提升团队生产率。

- 为了确保质量应该用精益的概念。每一小步确认限制级，确保达标。然后与客户确认，而不是先定一个总体的几个月计划。按总计划监控任务是否延误？因为后者会把团队的关注点都放到按时交付去，无法确保最终的产品达到高质量要求。
- 迭代回顾让团队可以每走一小步，回顾有那些不足，分析根因，下一步做改善。如果要从“救火”的管理思路变成基于根本原因找出预防措施的思路，就需要管理者‘放手’，让团队自己收集数据，自己分析与制定纠正措施。
- 收集数据很重要：
  - 收集数据应与迭代冲刺同步，除了收集团队内数据，也要收集客户的意见，才能全面看清问题
  - 如果把数据关联到绩效，很可能‘污染’数据，影响数据的正确性
  - 数据有显著变化时，便应更新基线，不要等到年底才更新

下一章，我们探索如何策划好迭代回顾与相关培训，让团队回顾后能提升质量和生产率。

## 附件

### 质量成本 COQ (Cost of Quality)

质量成本由三部分组成：

1. 失效 (Failure) 成本  
把缺陷修复好的成本，包括在客户现场被发现的缺陷。
2. 评测 (Appraisal) 成本  
包括各类测试，如系统测试，集成测试，单元测试等，所花的工时
3. 预防 (Prevention) 成本  
包括技术评审（注：有些人把评审归为评测成本，这里按 Mr.Juran 定义，归属于预防成本）

如变通理解以上 COQ 定义，“如何通过提高评审效率来降低质量成本”便可对应 COQ 各部分：

失效 (Failure) 成本：原本所有与缺陷相关的成本

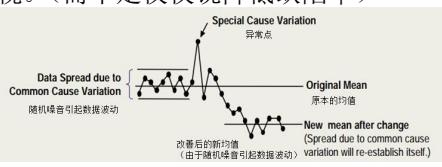
评测 (Appraisal) 成本：增加测试前的静态扫描、评审，减少失效 (Failure) 成本，使总 COQ 下降

预防 (Prevention) 成本：减小缺陷的产生，例如用原型做好需求，进一步使总 COQ 下降

## 公司高层不一定关注质量

问：在传统 IT 的公司，无论是缺陷 (Bug) 率还是其他质量指标对业务的影响的相关性都不容易度量，只有重大事件才会让公司在客户面前失去信任，所以高层不会真正的重视质量。

答：理解，但软件 BUG 的暴露越往后，返工的工作量就越高，而且是几何级别的增加（例如在单元测试或评审发现都可以 1 人小时内解决；系统测试通常要花起码 20 人时，客户使用后才发现便更高）。但在很多 IT 公司，大部分缺陷都是在系统测试、甚至到验收测试才发现。如果能把软件缺陷的发现前移，把通过系统测试、验收测试发现的缺陷减半，便可以大量降低质量成本，提升开发生产率。所以我建议利用缺陷数估算返工的工作量来引起老板重视。（而不是仅仅说降低缺陷率）



## 开发项目工作量（成本）分布

参考 Capers JONES 先生 2012 的例子，汇总成以下比例：

	30%	25%	20%	15%	10%
66% 测试与评审 100 (26%)					
13% 编码 83 (22%)					
21% 文档 66 (17%)					
	开会沟通 50 (13%)	项目管理 33 (8%)			
	发布后修改缺陷 20 (5%)				
	维护 32 (8%)				

测试与评审一般占开发工作量的 30%，测试与评审一般占总质量成本（包括发布后维护与改缺陷工作）的 66%，把所有工作量都加起来，测试与评审还是占最大 (26%)，编码第二 (22%)。

注意：测试与评审包括所有与缺陷相关的成本，包括单元测试、静态扫描、评审与相关的缺陷修正，而编码只包括设计与编写代码部分。例如有些人会觉得比例应该是开发 30%，测试和 bug 修改 25%，需求和设计 20%，项目管理和沟通 20%，文档 5%。

但如果按上面的定义，开发部分很可能已包括单元测试、静态扫描与改正缺陷工作，如把这些都归到测试评审里会变回类似上图的比例。



# Chapter 12

## 如何做好准备

### 分析事故根因：下集

我：陈总，听说你三年前开始锻炼马拉松，所以现在身体这么好，有什么心得可以让我学习一下？

总经理：本来我很胖，也常常很困，体检指标不太理想，我听说可以用长跑锻炼身体，便开始尝试。开始的时候不容易，因为一直都没有锻炼，先跑五六公里，按程序逐步提升，后面便慢慢养成习惯。最终经过三年锻炼完成马拉松，身体也觉得比以前好多了。

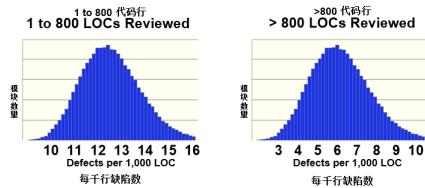
我：开始的时候，请问您怎么去持续维持，因为很多人还是一时冲动去锻炼，但持久不了，例如买了跑步机，但一两次后面就没再用了，您有什么心得可以让你持续下去？

总经理：现在支持长跑运动员的度量挺多，最简单的指标就是一公里要多少分钟。我就一直都有度量这个系数来监控进展。比如我开始的时候一公里要接近八分钟，跟走路差不多，后面慢慢就变成七分钟，六分半，六分钟。。。是锻炼出来的，所以有了这个数，我就知道自己现在是什么水平，是否在进步。

我：好比每天跑步锻炼，如果没有手表帮你度量时间速度，你是不知道是否有进步；数据可以给做事的人反馈，让当事人知道现在在差距有多少，所以如果可以从定性提升到定量管理，也可以产生同样的作用。例如，：如果评审发现缺陷数量太少，团队应该担心，因为很可能有不少缺陷未被发现，后面到了客户使用时才暴露，代价更大。感兴趣吗？

总经理：很感兴趣

我：首先须要利用以往项目数据，建立标杆（基线）例如下图：



如果代码是 800 行以下，每千行代码的缺陷密度均值与范围。800 行以上的缺陷密度会低一些，因分母较大。

如果某次代码评审，700 行的模块，代码评审发现的 2 个缺陷，比基线下限 7 个缺陷（注）低很多，团队便应担心，很可能存在不少缺陷未暴露。

总经理：但我希望团队有提升，不仅仅是维持，团队怎样可以利用度量数据帮助提升？

我：可以利用水晶球模拟，利用缺陷排除率，预估如使用新方法后的缺陷范围。如果也收集团队的缺陷返工工作量，我们更可以预估能降低多少返工。从降低后面缺陷返工入手，能提高产品开发质量，也能提高团队的生产效率，降低开放成本。

注：从基线左图，如按每千行代码有 10 个缺陷为下限，700 行代码便应发现起码 7 个缺陷

我用实例在白板上估算提前发现解决缺陷能降低研发成本接近一半。

总经理：很好，应怎么样开始？

我：你们有没有团队已经是按迭代开发？

总经理：有的，我们去年开始已经有些团队按一个月一个迭代交付，有些更短可能两到三周。

我：你们这些团队人员的能力主动性如何？

总经理：我觉得还是挺不错的。

我：如果你可以识别出两到三个正在在迭代开发的团队，就可以开始做量化的回顾，你们那些迭代团队以前有每轮迭代后做回顾的习惯吗？

总经理：都没有

我：首先就要让他们了解什么是根因分析？为什么要回顾/复盘？希望达到什么目的？你们管理层也需要有心理准备。开始的时候人员要花时间学习，因为你们也准备参加 CMMI 评估，团队做好量化迭代回顾就可以帮助你们从三级水平提升到 CMMI4 级，并开始利用项目迭代数据，建立基线和预测模型。但团队要做好第一次迭代回顾，就像要小孩学游泳，不能只是扔他到水池里面要他自己自学，必须先有培训帮助他。首先要安排这些团队参加根因分析和回顾培训，也需要你们内部的管理层包括有经验的主管作为内部教练，辅导团队，你们愿意尝试吗？

====

有了高层的支持，我们便可以试点，找团队做好迭代回顾，但不会是立马点石成金，是持续改进过程。

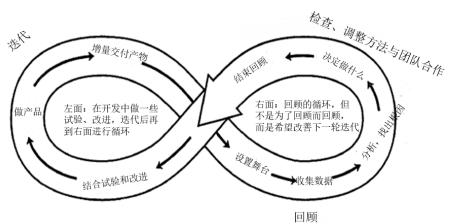
要利用根本原因分析做改进，必须有数据，也需要有机会立马实验改进方法，评判效果，所以迭代回顾（或复盘）是做根本原因分析的最佳时机。但很多团队只在迭代回顾时讨论如何解决迭代暴露的缺陷与相关职责分工，没有探索根本原因，和如何能避免同类问题再发生。

想了解如何做好回顾，先看回顾的主要步骤：

## 回顾流程

冲刺回顾可按下图的五步，确保各成员都全心投入参与，并能从分析形成行动，达到改进效果：

1. 设置舞台 -‘破冰’
2. 收集数据（除了收集‘硬’数据，如缺陷，工作量，进度偏差等，也要收集‘软’数据，如团员感受）
3. 分析，找出根因（除了鱼骨图，FMEA，也可参考附件里的 KJ 分析法，大家利用便利贴做分析）
4. 决定做什么（必须明确下个迭代的具体改进行动到人，任务，时间）
5. 结束



以上只是根因分析的技巧，若要团队做好迭代复盘，取得效果，还要注意以下原则：

### 做好回顾 4 原则

#### 1. 回顾让数据提供者能即时收到反馈

- 软件开发与工业生产不一样，大部分的数据必须开发人员自己收集，不能但靠机器/系统（例如：返工工作量）
- 如果回顾时分析根因是依据迭代数据，除了能帮助根因分析更具体外，也让团队成员有动力收集数据（因他们知道会一起分析数据）

因为软件开发是知识性工作，例如，某活动所花的工时，必须靠个人记录（具体步骤可参考‘个人量化管理’）

- 但记录数据要花精力，如果不了解收集的数据，后面有什么用途，便难以维持不断收集数据的习惯
- 反之，当大家都清楚到迭代回顾时会一起分析团队数据，并制定针对改进措施，就有动力继续迭代里统计数据

#### 2. 整个团队（所有相关干系人）参与

整个团队分析全局问题。例如，如果只是从测试人员的角度，他可能以为缺陷都是来自开发；需求分析人员（或产品经理）会觉得自己做得很好，因需求评审里没有什么问题被发现。但如果团队所有成员聚在一起（包括项目经理、需求、设计、编码、测试），把所有的缺陷列出来，让团队所有岗位一起分析，才可以全局看问题，才可能发现不少问题是因为需求没有明确，导致后面做出来

的功能并不是客户要的。后面可扩大参与回顾的干系人，例如客户代表，可以更全面分析。

### 3. 让团队自己寻找改进方案

“必须参与一起分析、讨论，才有动力后面采取行动”

“我们团队回顾都是全部成员参加，一起讨论”某团队组长说。以下是团队一起讨论后的根因分析结果：

项目组	原因
**雄	建议使用FindBugs代码扫描工具
张**	缺少系统业务培训
高**	没有合适的代码扫描工具
李*	希望公司制定缺陷数据分析标准
韩**	公司应提供合适的代码扫描工具
高*	公司应加强相关业务培训
*一铭	没有合适的代码扫描工具
孙**	建议使用有力的代码扫描工具

有没有看到组长提出建议使用代码扫描工具后，其他 7 位团队成员中 4 位也提类似原因？

所以当团队一起围着开会时，很多人都会避免提出与众不同意见，所以如果用投票方式，让大家对提出的“原因”进行投票时，票数最多的通常也是组长提出的方案（扫描工具）。

从 Asch 1951 实验（详见附件），看到人容易受群众压力影响（尤其当大家不认识），不敢独立表达个人看法，所以很多会议都只是项目经理一言堂，其他人只是观众，没有投入分析。为了鼓励每人畅所欲言，表达独立看法，并达到共识需要：

1. 鼓励每人用实例讲对质量、过程的观点/经验（寻找大家的共通点）
2. 鼓励每人分享个人感受，想法，满意开心/困惑的时刻（让大家相互了解）
  - 当大家感受到都是面对类似的问题，才会放开自己，分享个人看法
  - 教练的目标是让每人都有充分机会表达自己的观点

所以在回顾时，也要想办法避免发生这种心理上的影响，导致不能真正挖掘问题的原因。所以在回顾需要大家开放，以下两方法可以减少团队压力，让大家更投入，能畅所欲言：

1. 回顾开始时（设置舞台，“破冰”），用互动游戏，让团队放开顾虑，全心投入讨论（可参考附件里“游戏：应与否”）
2. KJ 分析法（详见附件）：每人都有一支水笔 + 便利贴，一起找根因，每人都可以写上自己的意见。而不是传统开会方式。很多时候都只是某人讲，其他人听。无法得到团队成员充分参与。如果可以给团队自主权，他们就

更能发挥、更能达到目标。

如果能让每位团队成员都有充分机会表达意见，不单能集思广益，做到更好的对应方案，也能提升团队后面执行改进方案的积极性，因为一般人都会更喜欢自己的选择。

详见附件里两个实验：

1. Brehm 1956 决策影响实验
2. 粮食分配实验

#### 4. 看全局，寻找大家的共同点

避免局部最优 (sub-optimization)，避免盲目摸象，最终希望可以全面从每个角色的视角全面分析。

某公司管理层一直非常关注度量，对各过程，包括需求、开发、测试等，都订了七八个 KPI 指标，并每月监控。但各个过程按指标做好不一定代表总体效果会好。每个过程都会影响软件开发最后遗漏到客户的缺陷数，迭代回顾的时候，团队可以用缺陷排除的预测模型让团队‘看见’如果代码评审排除率提升如何影响每个过程的缺陷数，不仅仅是定性的讨论分析。

例如，为什么要用‘设置舞台’开始迭代回顾？（例如‘应与否’是其中一种方法，详见附件），目的是让整个团队全心参与，如大家没有担心，愿意提意见，才能更全面看问题，寻找共同点，而不是辩解。

例如发现大量缺陷大部分都在系统测试和验收测试才暴露，大部分缺陷未在前面评审和单元测试发现并解除，根因分析讨论后发现单元测试没有任何要求，依赖程序员自己测。但很多时候程序员就因为时间压力都没认真去测。因为大家都忙，没有时间去深入去看，代码评审也没有发现多少问题。针对这些点大家觉得可以加强静态扫描工具，尽早发现一些语句基本问题，或重复代码等问题，减轻依赖人工查看；单元测试可以用脚本写，并规定有一定的语句覆盖率要求（例如， $>80\%$ ）。但如果团队只是按这些去做，没有量化目标就很难衡量，做得够不够？不然要等到迭代最后复盘时才知道效果如何。好比要设计一座新的 80 层高楼，建筑师预先用工程模型预估它防地震、防台风的能力，不要等到建好才知道有问题。团队可利用蒙地卡罗预测模型预先依据以往迭代的数据，把参数输入模型，比如每个阶段引入的缺陷数，每个阶段的缺陷排除率。首先可以利用蒙地卡罗模型，看看它出来的缺陷分布，验证是否对应以往迭代数据，预估范围是否可以覆盖以往数据。

第二步，就可以使用蒙地卡罗模型做一些实验：例如，我们估计用了静态扫描可以把代码评审的排除率从本来的 10% 提升到 50%，团队利用蒙地卡罗模型可以预估这种搭配的各个过程的缺陷分布，我们就可以看到如果这个排除率提升的话，评审应该出发现多少缺陷，比以往提升多少。团队也可以加入利用脚本的单元测试的缺陷排除率，例如估计能从本来的排除只有 20% 提升到 55%，看到对整个缺陷分布的变化？这样的话，我们就可以团队有下一轮迭代各个过程的缺陷预估范围参考。如果发现在代码扫描后这个缺陷数达不到本来的预期目标，便可立马采取纠正措施，不要等到最后迭代回顾时才发现未能达不到本来的预期效果。

如果有好几种方法选择，蒙地卡罗预测模型更可以帮我们做选择一个最优的最佳搭配，比如我们只是先做好评审，还是先做好单元测试，因为那些都会增加一些工作量，还是两边种方法都做，我们就可以要用蒙地卡罗以总成本最低选最佳搭配。反过来，如果没有这种蒙地卡罗工具的话，团队是很难看到每一种变化总体的效果，像盲人摸象。

因为工具可以反映过程之间的相互关系，如果我前面评审发现多了，遗漏到后面的缺陷就会减少，一层一层的关系，它利用那个缺陷排除的关系模型就可以估计总体的效果，也帮团队迭代回顾从定性改进分析，提升到定量。（如何从迭代的缺陷数据，利用蒙地卡罗预测模型，做定量分析，详见附件。）

## 团队迭代回顾根因分析常见问题

1. 没有利用缺陷数据做帕累托图分析，不清楚应针对哪一类，不清楚哪一类问题最多，导致根因分析讨论没有针对性。有些缺陷分类没有定义好，不明确或重同一缺陷可以归到几个类别，导致分析结果没有参考作用。
2. 不清楚什么是根因，只找到表面看到的现象，例如人经验不足，对技术不熟悉，对业务不熟悉等。没有抓到系统的根本问题。
3. 没有量化目标，只有定性的根因分析，下轮迭代难以判断效果能否达到。
4. 回顾时间不够
5. 设备、纸张、工具等未准备好

除了教团队根因分析技巧（定性）以外，也需要展示如何使用水晶球蒙提卡罗预测模型，预估各类缺陷数范围，有了范围才可以判断最终结果是否达标，差多远。识别出哪些过程有差异，下一轮回顾才能更有针对性。如果团队想下一个迭代降低遗漏到客户的缺陷数，尽量提前用评审或测试预先发现并解决。如果仅仅提一些改进方法，比如加强评审，使用工具等，效果很有限。例如团队一般也有做评审，例如需求评审，但无法判断评审质量如何？例如，发现四个缺陷是否足够呢？我们就可以用缺陷排除率配合蒙地卡罗模型，估计使用新提升方法后，预计的缺陷数范围，这样的话，如果团队发现缺陷数没达到预期，就知道立马可以加大力度加强，而不仅仅是走过程。（关于缺线排除率配合蒙地卡罗的原理，详见附件）

以上前三点可以利用培训加强相关能力，为了确保团队能做好第一次迭代回顾，建议要之前和团队做培训。后两点可以做好准备，预防问题发生，所以若要做好回顾，除了要理解上面 4 原则，也需要注意以下两条件：

- 回顾的环境，设备：有大白纸贴在墙上，水笔，便利贴等

有时便利贴可能太小，或字很小，原因是如果用一般的白板笔，字就太大了，如果用平常的 0.5mm 水笔，太细，字太小，都不合适。应该买一些比如 2MM 粗的水笔，才合适写便利贴。让大家可以在一到两米距离都能看得清。还有那些便利贴是不适合直接贴在普通白板上的，所以必须先从图文店买一卷比如两米宽的卷纸，用泥胶稳固在墙上；因便利贴只是上边有粘性，要在便利贴下部背面贴上一点泥胶，便不会出现便利贴下面翘起来，影响整个 KJ 报告看不清的情况。整个房间应该有充分的空间让大家活动，而不是那种传统一个大长座的会议室，因为越是让大家活动流动，才能投入。

- 足够的时间，起码 3 小时

某成都客户反馈研发总监：已经做了很多年过程量化考核，起初能给团队带来活力，因为至少比以前好，有路径可以量化他们，员工觉得做有所值，公司也看得见。前几年执行的很好，产出和上线频率比过去好很多。但是现在感觉逐渐拉不出差距来，因为大家都非常熟悉这套游戏规则。

我：理解，比如现在你们团队开始做迭代回顾，团队一起分析根因，持续改善。您觉得效果如何？

研发总监：挺好，但是还是感觉他们开回顾会，耗时太长了，2 个小时以上  
我：因需要团队收集数据，分析，讨论下迭代措施，所以通常要 3 小时，熟练后会快一点，底线是改进的节省应大于回顾的投入。

高层的支持是任何改进的重要成功要素。所以首先要让管理者赞同迭代回顾能为公司带回来回报（省钱），可以节省的成本（工作量）大于回顾的人力投入；也要让他理解任何改变都有个混沌的过渡期，要经过几轮回顾后，团队才能自己持续改善。

如果回顾只能一个或一个半小时的话，就不够时间让团队针对缺陷分组分类、画帕累托图做分析，没有根据实际数据分析根因，也影响到团队没有动力在后面迭代花精力去收集数据。例如，缺陷返工数据一直都非常难获取得到的，也需要在迭代根因分析时给时间团队讨论统计，所以通常一个完整的迭代根因分析回顾不会小于三小时。要做好这块如果有本地的教练可以更好控制整个回顾的节奏。不会在某一个环节耽误太多时间，也不会太长，也确保大家团队人员都全程投入讨论参与。这个我们在下一章会详细讲。

## 培训：策划与准备

培训能让团队能先了解利用模拟工具量化管理的原理，也让团队为收集迭代数据做准备，所以建议先做模拟互动培训。

培训的目的：

1. 让团队用模拟数据体验一个迭代回顾量化的应如何进行。
2. 知道为什么要收集缺陷和返工工作量的用途。并利用类似一般开发项目的环境的缺陷数据模拟。
3. 收集到团队真实数据是最大的挑战，培训过程中也要团队自己讨论准备如何收集迭代的数据。因为没有数据是无法做量化的迭代回顾。

下面是一天互动培训的时间安排：（上午是做上一章迭代回顾的互动练习，下午是使用蒙特卡洛预测模型预测下迭代的互动练习。）

如何提升软件开发质量Training Plan	
9:30	用案例介绍根因分析的五个步骤 与技巧：帕累托图 鱼骨图 等
10:00	根因分析小组互动练习
11:00	Break
11:15	迭代回顾场景（由公司内部教练做校色扮演）
11:45	总结，并介绍缺陷排除率模型
12:00	午休
13:00	介绍缺陷排除率模型
13:15	量化根因分析模拟练习，包括： 分析模拟数据，估计缺陷排除率与返工工作量 针对主问题，团队所有成员做根本原因分析 从根因、讨论对应改进措施
15:15	Break
15:30	设定量化质量计划模拟练习 利用MonteCarlo工具 对原本迭代数据做3点估算 再依据前面制定的方案 利用工具寻找最佳方案
16:15	总结，确认团队后面的具体工作

## 时间安排

如果是首次根本原因分析培训，建议先在培训当天早上做以下根因分析小组互动练习，让大家先熟悉根因分析的重点：

- 提供一对模拟数据，要求团队利用帕累托图加鱼骨图分析，找出根本原因与改进措施
- 目的：让团队先感受如何基于数据，找出根本原因

## 培训对象

- 后面会参与试点的项目组，培训后可以把学过的在项目里实践

- 所有团队角色都参加

准备数据，让团队模拟

要做量化根本原因分析就必须要有数据，所以内部教练须要预先准备下面缺陷与返工工作量数据，类似一轮冲刺后的场景：

- 系统测试缺陷数据，建议从缺陷管理工具里抽取，40 - 60 项
  - 开发个人系统测试 BUG 返工工时统计表：那个 BUG 号 / 总修复工时 / 备注
  - 开发个人开发期间用于修复评审/单元测试返工工时统计表：那个模块 / 总修复工时 / 备注
- 打印以上的数据表，发给团队做练习时参考

例如，互动练习时，要学员利用开发人员工时表（模拟他们迭代中用于改缺陷的工时，与评审缺陷数，与开发和修正代码的工时）：

缺陷 ID	系统测试bug描述	开始	结束	总修复工时	备注
5384	当客户选择“领（非大陆身份证件）”时， 大路地址留空，系统报错，无法继续 输入。输入框内显示“领”	17:30-17:30	17:30-17:30	10	技术资源匮乏，修复困难， 分拆需求待议
5198	输入错误结果日期，但系统没有报错。 (估计程序没有将输入是否正确)	17:00-17:00	17:00-17:00	25	
5365	输入-9.99 查询，结果是阳性，不对。 输入-9.99 查询，结果是阳性，不对。	17:30-17:30	17:30-17:30	5	内部压力，急需修复

开发功能	开始	结束	人时	缺陷	修改	LOC	USS	备注
1) DISPLAYMAP	9:00-12:30	12:30-12:30	6	售后软件 不能兼容新规则				
2) SORTCUST	14:30-17:30	17:30-17:30	9	待处理需求				
3) LOGIN	17:30-17:30	17:30-17:30	3	需求， 需求变更没跟上				
4) FINDMEDIAN	17:30-17:30	17:30-17:30	4	误解语言				
5)								

准备工具

- 分 2-4 组，6-8 人一组，需要有一个大而光亮的会议室，方便小组互动，大白板和架子，各种颜色的水笔，墙壁可以贴上小组的白纸
- 教练提供有 MonteCarlo 工具的笔记本电脑

## 使用二八原则与 KJ 分析根因实例

公司背景

多年来，这家公司一直做医疗 IT 解决方案，门槛很高，不仅要懂 IT，也要懂行业知识。医院，或医疗机构，都对质量有要求。

工作压力也很大——去年业务扩展很快，除了要增加软件产品数量以外，也非常注意产品质量。例如，公司用系统统计客户反馈的缺陷。要求过程改进小组，按月统计客户反馈的缺陷，看是如何逐步到得到改进、不断完善。因员工人数快速增加，公司也注重控制成本，如研发成本。

从项目冲刺的回顾复盘开始

质量经理把我引到回顾复盘现场，技术总监（高层）也参加，测试人员投影了本次迭代的缺陷分析，展示下面两个缺陷分布图：

- 开发人员排名（最多的排头）
- 按模块来区分（最多的排头）

团队组长便按照缺陷的严重程度询问每位开发人员 -- 是否知道怎么修改？大家都说知道了。组长要求大家提出问题，如果没有问题就准备散会。

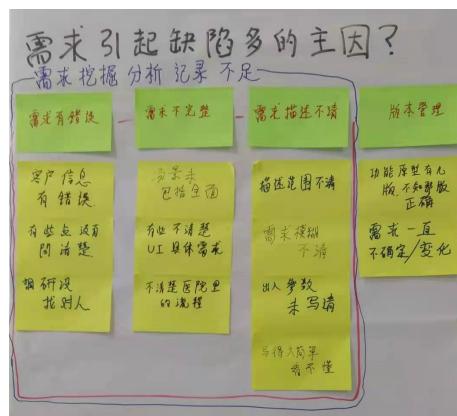
我问：大家好，你们有没有兴趣做个小实验。后面让我们暂时抛开自己本身是什么岗位。大家一起看看有哪些地方能减少缺陷？

例如，除了按照人员与模块区分。我们可否把缺陷简单按下面类型分组：需求/设计/编码？

他们就让测试人员，展示本冲刺发现的 34 个缺陷，让大家逐一讨论，找出缺陷是源自哪过程，最后把总数写在白板上：

需求	19	56%
代码	9	26%
设计	2	6%
其他	4	12%
34		

让我们看看是怎么分布？为什么这类缺陷最多呢？大家估计背后是什么原因？针对需求（最多的类别），我辅导团队利用 KJ 方法 [详见附件]，识别主因。最终汇总成以下结果：



### 分析问题 - 根因 - 行动

我：针对我们刚才一起找出的主因，根本原因是什么？如何可以避免同类缺陷再发生？

总监：要加强对需求人员的培训，提高他们的能力。

我年初已察觉到确实很多需求没有表达清楚。导致开发出来的东西不符合客户要求，或不是客户要的。

总监接下来说：

我见过以下用户故事：

病人或他的家属能容易找到他选择的服务。

理由：他们熟悉网上购物，习惯了方便和快速的响应时间。

我问需求人员怎样才算容易找到，验收标准？

我估计她记得我说过需求必须可测量，她想了一会，说：验收标准：“普通病人能够在 6 秒钟内通过不超过三个动作定位任意一项服务。”

我说如果把‘普通病人’改为‘90% 以上的病人’更好。

必须把模糊，有二义性的需求变成可以测量。

所以不能仅仅说‘新功能很酷，很创新’，而应明确验收标准为：引入了新功能的三个月之内，60% 的用户应该用它来完成规定的工作。75% 以上的用户对产品表示赞许。

所以三个月前，我已经开始准备正式规划产品经理与需求分析师岗位：要经过挑选，考试，然后培训，达标才能正式上岗。本来计划两周后会正式公告，现在既然你们问到，我就预告一下。

我：既然高层已经有长远的规划，我们团队就应该针对下一个冲刺，我们可以做到的事情。

团队：我们每个岗位都已经尽了全力，没有什么可以做了。

我：你们有做评审吗？如需求，设计评审

团队：有。

我：需求评审发现多少缺陷，设计评审发现多少？

团队：好像两三个。

我：发现什么问题？

团队：记不清了，当时直接就修改了。

我：请问评审总共花了多少时间？多少人参加？

团队：我们六个人，那次评审大概用了接近 2 小时。

我：2 小时？

团队：我们不仅仅发现需求问题，也一起讨论如何修改

我：如果评审只找缺陷，记录，应不会超过一小时，如果大家事前做好准备，估计可能半小时可以完成。所以通常检查（Inspection）不会当场讨论如何修改。

我接着说：我们刚才分析系统测试缺陷，不是识别出超过一半是源自需求吗？为什么我们不能在评审时预先发现？你们觉得可以下一个冲刺，评审时可以发现更多缺陷吗？

我立马用 5 分钟与大家分享有效评审能提高产品质量，降低成本的例子。

团队：估计应该可以，但不知道怎么做？

我：你们评审有检查清单吗？

团队：没有。

我：清单可以帮助我们吸收以往的经验，避免以后同类问题再发生。例如刚才我们都识别了跟需求挖掘/分析/记录不足相关的问题吗？可否利用这些，更新评审检查单的检查项项，提醒我们要避免同类问题。如果大家同意，我们现在就行动。我们要改进便要制定目标，例如计划下次需求评审，系统测试等各过程发现的缺陷数。这些目标你们可以下周一策划两周冲刺时定。

组长安排了小李更新检查单。准备在下次评审前与需求文档，预先发给参评人员。

我：谢谢大家，我没有其他要说了，下次回顾，我或总监会来参加，看看冲刺的效果。

要利用根本原因分析做改进，必须有数据，也需要有机会立马实验改进方法，评判效果，所以迭代回顾（或复盘）是做根本原因分析的最佳时机。但很多团队只在迭代回顾时讨论如何解决迭代暴露的缺陷与相关职责分工，没有探索根本原因，和如何能避免同类问题再发生。

## 经验教训

很多开发人员虽然编码/测试很有经验，但不熟识统计分析，蒙地卡罗模拟等，所以整个培训需要有内部的教练辅助他们怎么利用那些工具技巧去分析数据。团队只是提供数据和分析结果，不一定要很了解里面的数学分析，但必须真正了解根因分析，所以我们培训会以根因分析开始，用前面的丰田故事，日本绅士俱乐部原因分析故事。让大家了解根本原因分析的重点。然后给小组前面的机场延误数据，让他们利用 KJ 模拟方式和帕累托图，自己动脑筋找出根因，我们的经验：越多互动练习，他们就越感兴趣，越主动参与。所以如果时间有限，尽量可以压缩讲理论的部分，甚至要他自己先阅读了解，大部分时间让他利用数据分析根因。有了这个基础就可以进入第二部分，利用内部教练准备好的几十条系统测试缺陷，要他们按缺陷排除的方式，找出缺陷的主要来源，并估计返工工作量，再利用蒙地卡罗模拟估计每一个过程的缺陷范围。经过培训，他们便可以在下一个迭代回顾时用同样思路，分析自己的迭代数据，找出根因并制定下一轮迭代的纠正措施。培训结束前，内部教练在总结时要求大家团队讨论后面有什么方式可以收集到做迭代回顾所需要的数据，例如缺陷数，返工工作量等。

## 结束语

要做好迭代回顾，首先必须得到高层的支持，并取得所需的资源与时间。策划与培训是成功的第一步。因一般团队没有回顾与根因分析的概念，教练如何现场辅导也很重要。

下章会分享如何在回顾中的注意事项和如何持续。

## 附件

### 游戏：应与否

在迭代回顾中使用。

### 目的

帮助建立有效沟通的思维模式。帮助参与者抛开指责和判断，以及对指责和判断的恐惧。

### 所需的时间

8 ~ 12 分钟，取决于小组的大小。

### 描述

在描述这些模式之后，参与者讨论这些沟通模式的意义。

### 步骤

1. 将注意力集中在“应与否”(见下图)，并简要阅读。
2. 组成小组，每组不超过 4 人。要求每组用一对词来定义和描述。如果有四对以上的词对/组，如果有多个组有相同的词对，也可以。
3. 让每个小组讨论他们的两个单词的意思和它们所代表的行为。让他们描述各自对团队和回顾的影响。
4. 每个小组向整个小组汇报他们的讨论情况。
5. 询问大家是否愿意使用左面的方式。

500px



### 用角色扮演为做好回顾打基础

学员用各种角色扮演什么是吵架，什么是交流，让观众和表演者更能感受到量化回顾先要有对事不对人的心态：



辩解

### 如何分析缺陷数据得出预测模型参数，出分布图

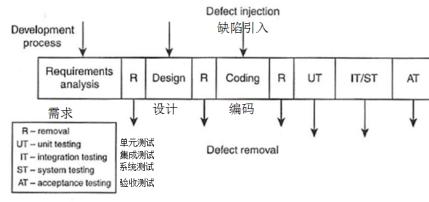


Figure 7.1 Defect injection and removal

需求、设计、编码后都会评审/测试来排除缺陷，但仅仅做评审/测试不一定能确保质量。因为最终验收缺陷数取决于每个步骤能否有效排除当前的缺陷。

可以用缺陷排除率 (Defect Removal Efficiency DRE) 来衡量测试或评审的效率：

$$DRE = \frac{\text{Defects found by the QC activity}}{\text{Total defects in the product before QC}} \times 100\%$$

有些人会认为尽早发现并解决缺陷

对质量肯定好，但会耗费工作量，增加项目成本，老板不一定愿意。

其实是反过来，如能在前面预先发现并修正缺陷，

便能减小后面测试和修改缺陷的工作量，

最终只会减少总项目工作量。

A. 只依赖测试				B. 加上评审和扫描预防缺陷阶段 且增加测试排除缺陷效率 9%				
系统中缺陷总数	1000			1000				
排除缺陷效率:	固定	变动	总工时	固定	变动	总工时		
测试前排除缺陷	0%	1000		20%	800	25	100	
静态扫描 审查/拦截编程	0%	1000		80%	200	25	240	
测试/修正缺陷	25%	750	25	250	35%	254	25	95
单元测试	25%	548	25	203	33%	150	25	74
功能测试	25%	411	25	137	30%	108	25	46
回归测试	35%	287	25	144	40%	63	25	45
集成测试	15%	227	25	40	15%	54	25	30
最终测试		125	773	898	175	806	781	
最终总缺陷数 缺陷消除率	227	77.3%		54	94.8%			

比较以上两种策略的质量成本 (COQ) 就能看出：

- 增加测试前扫描与审查，并加大测试效率，不仅减少最终缺陷数到 54 (对比 227)，也降低总质量成本 (总人时)

假设：每项任务的固定成本都是 25 人时；测试前的缺陷修复：每缺陷用 0.5 人时；测试缺陷修复，上面计算假定每缺陷用 1 人时，详见下表：

每缺陷修正工时	
扫描/评审	0.5
单元/功能测试	1
系统/性能测试	1
验收测试	1

- 通常测试里的缺陷修复每缺陷不只一人时，例如，在验收阶段时可能要用 20 人时；我通常会打开这 XLS 表，填上客户的估计值。从上表看到，即使用最少的 1 人时来算，还是不亏。

因为缺陷已经提前被排除系统测试的缺陷减少，测试阶段的工作量减少大于前面扫描评审上的投入。

可以进一步利用 COQ 概念，增加早期预防缺陷措施，如用原型与客户交流，做好需求调研，进一步减少缺陷，和成本。例如，使用原型与场景与客户挖掘需求，可进一步把缺陷降到 43，也降低总质量成本：

		测试加上评审和扫描预先排除缺陷， 也增加测试力度				加上利用原型场景避免缺陷				
		1000				1000				
避免缺陷	原型/场景	排除缺陷效率：				排除缺陷效率：				
		20%	800	25	100	20%	640	25	80	
测试前排除缺陷	静态扫描 审查/结对编程	60%	320	25	240	60%	256	25	192	
测试/修正缺陷	单元测试 功能测试 回归测试 系统/性能测试 验收测试	30% 33% 30% 40% 15%	224 150 105 63 54	25 25 25 25 25	96 74 45 42 9	30% 33% 30% 40% 15%	179 120 84 50 43	25 25 25 25 25	77 59 36 34 8	
最终遗漏缺陷数 缺陷排除率		94.6%	54	175	606	781	43	200	485	685
		95.7%								

注：你可能会质疑使用原型方法不只 25 人时，但即使加大到 100 人时，还是不亏。因它能预防缺陷，整体缺陷数下降 20%，使总质量成本下降超过 100 人时。

下面我们介绍一下怎么从引进量化管理来提高开发质量：

# 设定量化目标。例如希望最终遗漏到验收发现的缺陷数降多少？

1. 并设定中间阶段目标缺陷数，从而预测能否达到最终目标，不要等到最后才知道不满足

可用缺陷排除率来判断每一个过程的质量：

如果要最终质量好，缺陷排除率就要高。但计算缺陷排除率，必须要等到整个开发完成才可以计算，我们可以建立一个预测模型，模拟这个过程：

需求会引入缺陷，然后需求评审排除缺陷等等。把各引入缺陷数，排除率输入蒙地卡罗预测模型，然后使用它估计每个阶段的缺陷数量范围。

Table 7.2 Defect Distribution in Infosys's PCB Infosys PCB 的缺陷分布

Process Stage 过程阶段	Percentage of Total Defects 占总缺陷数百分比
RR + DR Requirements review 需求评审 + HLD + detailed design review 需要/详细设计评审	15% (15% - 20%)
CR+UT Code reviews 代码评审 + unit testing 单元测试	55% (50% - 70%)
IT+ST Integration testing 集成测试 + system testing 系统测试	22% (20% - 28%)
AT Acceptance testing 验收测试	8% (5% - 10%)

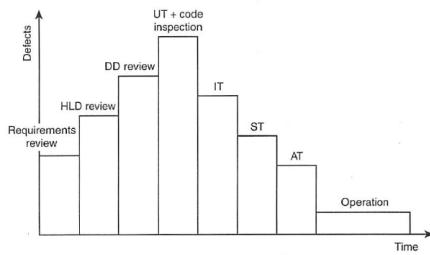
如果我们按上面 infosys 公司的各阶段缺陷利率百分比，设计与需求排除缺陷 15%，代码评审与单元测试 55%，集成/系统测试 22%，验收测试 8%，下面按缺陷总数为 100，得出算出设计与需求排除共 15 缺陷，DR+RR=15，CR+UT=55，IT+ST=22，AT=8，求出各阶段的缺陷输入与缺陷排除率，把参数输入蒙地卡罗预测模型：

表 D1:

	Req 需求	Design 设计	Coding 编码
RR	5		
DR	5	5	
CR	8	5	10
UT		6	26
ST		5	17
AT	2	1	5
总数:	20	22	58

	D.R.E.
需求评审 RR DRE = 5/20	25.00%
设计评审 DR DRE = (5+5) / (20+22-5)	27.03%
代码评审 CR DRE = (8+5+10) / (100-5-5-5)	27.06%
单元测试 UT DRE = (6+26) / (100-5-5-5-8-5-10)	51.61%
系统测试 ST DRE = (5+17) / (100-5-5-5-8-5-10-6-26)	73.33%
验收测试 AT DRE = (2+1+5) / (100-5-5-5-8-5-10-6-26-5-17)	100.00%

得出下面 Figure7.2, 缺陷分布是中间最高头尾低, 右面与左面不同, 有条长尾巴, 类似估算软件开发项目工作量的 Rayleigh 曲线。



**Figure 7.2** Defect detection in a project

模型假定每个阶段的缺陷排除率都比较稳定，在某个范围之内，不同阶段引入的缺陷也在一定范围之内。蒙地卡罗模型让我们可以设定缺陷排除率和缺陷输入数量的波动范围，预测各阶段缺陷排除分布范围，不仅仅看单点数（前面在第三章里介绍过如何使用蒙地卡罗模型做三点估算）。

## 使用水晶球蒙地卡罗模型实例

- 在模型参数输入部分输入估计对应缺陷排除百分比的上下限与均值，工作量也类似。例如，代码走查的缺陷排除率是 55

- 模型会自动按输入的最大最小值用 PERT 方程式计算标准差，变成分布（正态（normal）分布、），水晶球模型会依据 Decision 格的数字选取对应的分布，

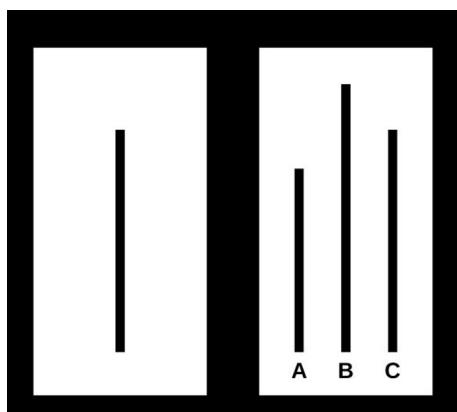
	Decision	1 Traditional Spec Q2 KJ Analysis & QFD	2 Prototyping	Zone 3 V2: AC60
RR Decision	2	std dev simulation std dev simulation std dev simulation	std dev simulation std dev simulation std dev simulation	std dev simulation std dev simulation std dev simulation
需求评审	3.75722634	3.33333333	3.33333333	3.33333333
Design Decision	58.477171	0.05	0.016657	0.016657
设计	55.731568	3.33333333	3.33333333	3.33333333
DR Decision	2	1 SA/SD	2 OOD	3
设计评审	22.534017	3.33333333	3.33333333	3.33333333
Code Decision	2	1 Email Routing	2 Walkthrough	3 Inspections
编码	238.995	0.05	0.016657	0.016657
CR Decision	2	1 Manual w/o Reuse	2 Manual w/Reuse	3 Code Generation w/o Reuse
代码评审	19.017144	3.33333333	3.33333333	3.33333333
UT Decision	2	1 Ad Hoc	2 Path Testing Only	3 Data Flow Testing 4 Both Path and DFT
单元测试	57.741915	3.33333333	16.66666667	16.66666667
	147.916115	3.33333333	3.33333333	0.00833333

- 成本参数：例如 Quality 行：修复一个系统测试发现的缺陷，平均要用 8000 元，最高 9200，最低 6800。例如 Effort 行：平均人时成本，平均是 600 元，最 750，最低 500。水晶球模型就会依据输入依据三角形分布估算单位成本的分布，放在绿格里。

	Costs	Unit Costs	Min	Most Likely	Max
Code 编码	Effort \$226,846	Quality \$500	\$700	\$800	\$1,000
Code Review 代码评审	Effort \$17,917	Quality \$219,686	\$700	\$800	\$1,000
Unit Test 单元测试	Effort \$280,774	Quality \$175,983	\$700	\$800	\$1,000
Integration Test 集成测试	Effort \$1	Quality \$5	\$700	\$800	\$1,000
System Test 系统测试	Effort \$64,207	Quality \$113,610	\$700	\$800	\$1,000
Acceptance Test 验收测试	Effort \$23,336	Quality \$49,306	\$700	\$800	\$1,000

- 估算质量成本：水晶球会依据每过程的缺陷分布和单位成本分布，预估质量成本的分布。例如这例子是需求评审用审查 (2)，设计评审用走查 (2)，代码评审也用走查 (2)，单元测试是手工 (1)，系统测试估计走 3 轮 (2)，验收测试预计走 3 轮 (2) 的预估分布结果。(括号里的数字代表在模型参数表里那一栏，从 1 至 4。如想了解水晶球如何把每个过程“加”起来，详见‘三点估算’附件：蒙特卡洛模拟)
- 模型经过 10,000 次模拟得出质量成本分布，和过程的缺陷分布与范围 (90% 置信区间)

### Asch 1951 群众压力实验



问题：请问右面那条线 (A, B, C) 的长度最接近左面线的长度？



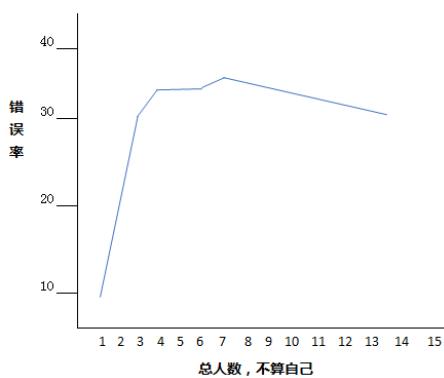
实验设计：随机邀请几位志愿者（包括你）参加，实验之前与（除了你以外）所有人预先说好，都选 A，并且要假装成经过深思熟虑。实验时，你是最后一位作答。

结果：

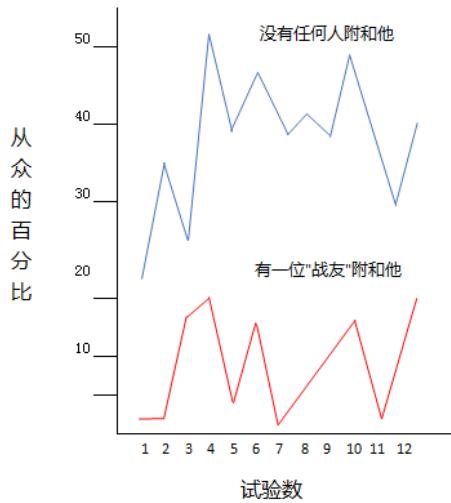
- 有群众压力时，大概 **37%** 会从众选择错误答案（虽然不同人有差异，但总体只有 **25%** 能坚持自己的正确答案。）
- 个人自己作答，接近 100% 选对，错误率少于 1%

影响因素：

- 前面选择错误答案的人数越多，错误率越高：
  1. 一位：接近 0% 答错
  2. 两位：大概 14%
  3. 三位：32%（详见下图）



- 如果前面有一位“战友”选了正确答案，你的错误率就显著降低（大约原本水平的  $1/4$ ，看下图黄线），并让你感到与他更亲切



- 尴尬场景 (例如迟到), 你的错误率会更高。
  - 实际你没有迟到, 只是让你觉得确实迟到了, 觉得尴尬。
- 如果不是要你说出答案, 而是让你把答案写在纸上, 错误率会降低三分之一。

## Brehm 1956 决策影响实验

实验设计:

1. 提供十多种不同礼物, 包括台灯, 多士炉, 挂钟、收音机等。
2. 请你按自己喜好对礼物排序。例如最喜欢的选一, 如果同样喜欢两件礼物, 可以不分高低, 写同一个数字。
3. 让你从两件同样喜好的礼物, 让你选其中一件, 让你拿走。
4. 但在你离开之前, 请你再对所有礼物按喜好排序。

结果:

- 绝大部分人都会抬高自己选择拿走的那一件礼物排序, 调低非选择的那件礼物排序。
- 但如果不是自己选择, 而是由老师选好给你, 你后面的礼物排序就没有变化。

结论:

- 人都会觉得自己的选择比较好。
- 例如选大学、选汽车、选伴侣。如果是由你自己决定选择 (非被安排), 你会觉得自己的选择较好。

## 粮食分配实验

二战时, 美国虽然不是战区, 也需要控制粮食供应。政府面对的难题:

- 怎么让那些家庭的消费习惯，满足战时的食物供应。

比如当时有些食物是要分配的，如何让家庭多吃一些供应充足的食物，避开紧缺的食物。他们实验发现，首先要知道整个过程是家庭里哪个人做这一块的决定。

研究分析发现，绝大部分的家庭都是由家庭主妇决定购买什么、储存什么、怎么做菜，丈夫没有太多的意见，通常是由媳妇决定。

实验设计：

- 把家庭主妇分成两组：

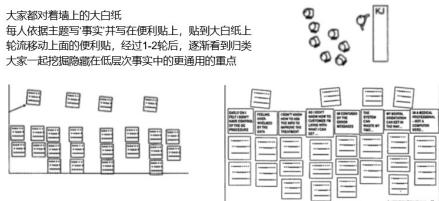
1. 专家跟一组家庭主妇宣扬某种食物营养丰富，对家人的身体都会有好处。
2. 另一组家庭主妇没有专家指导，只是给她一些营养的数据，邀请主妇分成小组自己讨论决定怎么去做。

结果：实验发现第一种方法没有效果，第二种效果却很好。

## KJ 分析方法步骤

让团队了解大家的事实，  
一起分析根因，一起做分析报告

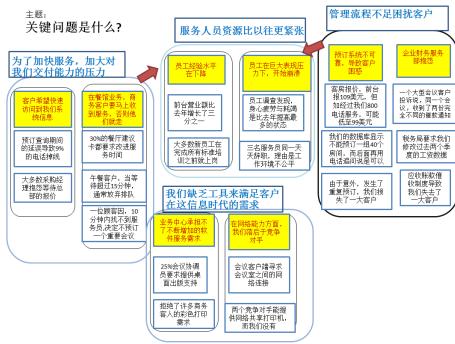
- 大家都对着墙上的大白纸
- 每人依据主题写事实并写在便利贴上，贴到大白纸上
- 轮流移动上面的便利贴，经过1-2轮后，逐渐看到归类
- 大家一起挖掘隐藏在低层次事实中的更通用的重点



一种头脑风暴方法，每人把自己想法写在便利贴上，一起轮流把所有便利贴排放分组



1. 把主题以问题形式写在大白纸的头顶
2. 把相关的事写在便利贴上面，用黑色
3. 搜集相关的事，如果是一组人的话，分散到不同人手上
4. (团队) 查看描述是否不清楚，是否要整理？
5. 每人轮流把相关的便利贴组合在一起
6. 当每人都已经调整过便利贴组合后，一起把每一组便利贴加上一个题目，用红色标识
7. 红色的标题下包含相关事实的内容
8. 把相关的红色题目(最好不超过3个)组合在一起
9. 给这大组一个题目 - 蓝色
10. 在题目下放上相关的红色小组
11. 把蓝色的题目和剩下红色的小组或单独没有分类的便利贴都在墙上放好，也可以用一些箭头描述它们的因果关系



## 水晶球蒙特卡洛预测模型

- 在模型参数输入部分输入估计对应缺陷排除百分比的上下限与均值，工作量也类似。例如，代码走查的缺陷排除率是 55

- 模型会自动按输入的最大最小值计算标准差，变成分布（如：正态（normal）分布、三角形（triangular）分布），水晶球模型会依据黄格里的数字选取对应的分布，

- 成本参数：例如 Quality 行：修复一个系统测试发现的缺陷，平均要用 8000 元，最高 9200，最低 6800。例如 Effort 行：平均人时成本，平均是 600 元，最 750，最低 500。水晶球模型就会依据输入依据三角形分布估算单位成本的分布，放在绿格里。

Code 编码	Costs		Unit Costs	Min	Most Likely	Max
	Effort	\$226,846	\$501	\$700	\$800	\$1,000
Code Review 代码评审	Effort	\$17,817	\$172	\$700	\$800	\$1,000
Code Review 代码评审	Quality	\$215,688	\$15,025	\$1,800	\$2,000	\$2,200
Unit Test 单元测试	Effort	\$280,774	\$800	\$700	\$800	\$1,000
Unit Test 单元测试	Quality	\$175,563	\$9,422	\$2,300	\$2,400	\$2,600
Integration Test 集成测试	Effort	\$1	\$542	\$700	\$800	\$1,000
Integration Test 集成测试	Quality	\$5	\$3,975	\$3,000	\$3,500	\$4,500
System Test 系统测试	Effort	\$64,307	\$786	\$500	\$600	\$750
System Test 系统测试	Quality	\$113,610	\$8,074	\$6,800	\$8,000	\$9,200
Acceptance Test 验收测试	Effort	\$23,336	\$768	\$700	\$800	\$1,000
Acceptance Test 验收测试	Quality	\$49,906	\$16,504	\$12,000	\$15,000	\$18,000

- 估算质量成本：水晶球会依据质每过程的缺陷分布和单位成本分布，预估质量成本的分布。例如这例子是需求评审用审查（2），设计评审用走查（2），代码评审也用走查（2），单元测试是手工（1），系统测试估计走 3 轮（2），验收测试预计走 3 轮（2）的预估分布结果。（括号里的数字代表在模型参数表里那一栏，从 1 至 4。如想了解水晶球如何把每个过程“加”起来，详见‘三点估算’附件：蒙特卡洛模拟）
- 模型经过 10,000 次模拟得出质量成本分布，和过程的缺陷分布与范围（90% 置信区间）

# Chapter 13

## 实施与维护

### 准备迭代回顾 Q&A

两个月前，为北京某企业做差距分析，发现大量缺陷在系统测试才被发现，建议加强迭代回顾根因分析。质量经理便开始尝试内部推动，并在两周前做了一次团队回顾辅导。

### 做好现场辅导

质量经理：我们计划下下周三就会与另一项目组做迭代回顾，他们团队共 7-8 人，2 周前与另一团队使用便利贴做根因分析，确实能听到更多不同的声音。应如何准备好，让今次这团队能放开并做好根因分析？

我：让我分享一下我的相关经验：

如果他们没有学过根因分析，应先用 15 分钟介绍根因分析的主要元素（可用绅士俱乐部案例），然后立马请他们分两组，使用机场延误数据做 45 分钟互动练习。（因利用数据，动脑筋，是最佳学习方式。因大家都坐过飞机，也遇过延误，应能理解机场的大概情况，所以不需要用实际团队缺陷数据。）

练习过程中，我只会定时提醒时间（需要在 45 分钟后跟管理层汇报），有时候在做练习时，学生会问我，“我做对不对啊？是否做错？”除非他们犯了基础错误（例如，用电脑打帕累托图，或只是按延误数据被一条按数字从最大排到最少等错误），我不会说你应该怎么做，尽量让团队自己讨论分析。团队都做完汇报以后，我才总结有哪些误解，那些做得好。要做好回顾，准备很重要，例如要用大白纸，要有足够空间。如果团队成员都是围着一个大桌，坐下来讨论，不会有好结果。但反过来，如果我们在墙上贴上大白纸（1.5 米 x 3 米），并提供便利贴，大号、小号水笔，每个人有自己水笔写的时候，他们就会一直在走动，讨论，写字，画图，并不断交换位置，互相帮忙，这样就更能体现他们的团队意识更投入。



（时间管理很重要。电脑投屏显示每轮活动剩下的时间（分钟），让团队即时知道还有多少时间。例如最后时间到会显示“TIME'S UP”）

机场飞机延误练习有几个很重要的特点：

- 真实的数据让团队可以按练习依据数据去分析根因，而不是纯粹头脑风暴，定性分析。
- 有明确的任务目标（45 分钟后给机场管理层汇报），给团队时间压力，团队便更有动力。只要任务明确，团队人员通常都有足够能力管理好自己分工，完成目标。

所以导师的任务不是教他一步一步怎么做，只要提醒时间，先定好目标，让他们团队自己发挥，自己讨论效果会更好。只要他们清楚需要做什么，大部分团队都能自己分好工，例如有些人画帕累托图，有些人做总体根因分析，或画鱼骨图，他们整个团队可能有五到八个人，我们作为导师就不需要帮他预先分工，他们自己会讨论自己会做好分工更好，尤其是已经是熟悉的、合作过的团队。

质量经理：你的意思是要减少干预，尽量让团队自己动脑筋。但因未做过，可否再举些例子？

我：不如我反过来，建议你作为导师应该避免说什么：

1. 其实你们团队我见到的其他团队好或者差（因这都是个人观点、感觉，非事实）
2. 本来只需要你们画帕累托图和鱼骨图，为什么你们还画了个脑图？
3. 你们都已经讨论了很多细节，要立马赶紧时间做最后总结
4. 你们讨论好像都只是针对机场闸口的问题，是否应该也考虑其他？

为什么要鼓励团队表达不同意见？

如果团队要改进，首先应让大家充分理解各成员的差异，充分讨论后大家便能自己找出共通点，才有机会一起改进。

如果大家可以在回顾时开放讨论，把自己的具体看法说出来，就可以让大家看到全貌，真正的情况。

所以内部教练尽量要鼓励不同的声音发出来，ASCH 1951“从众实验”证明，只要有一位战友，团队成员就愿意表达自己意见。所以内部顾问尽量要让每个人都有充分机会发声，越多不同的意见提出来，大家越能了解真正的情况，找出一个大家都赞同的改进方案。（详见附件：功能性分组帮助团队成长实例）

例如，如果发现团队一起开大会，因人多，不愿意发言，可能要分成小组讨论，效果更好。

有些团队比较“慢热”，便需要给他们多点时间，不怕沉默，等一下。只要等到有第一个人开始发言，后面就好办理了。

## 分析全过程

质量经理：好的，你说我们上次做了回顾还没有找到真正根因，但不知道如何改善。

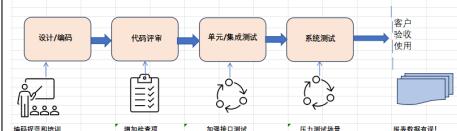
我：可考虑分析流程。

某家电力服务公司的内部软件开发团队，公司规定每两周固定日期发新版，因客户对软件质量要求很高，每次都必须通过一系列的评审和测试才可以发版。如果不能按发版时间完成项目交付，经理和团队会复盘整个过程，以识别在整个过程里哪些地方出现了失效点，然后针对失效点分析根因。所以除了模型，流程图也可以帮助分析过程并找出根因。画了流程图后，便可以利用 FMEA 方法针对每个失效点，找根因，和对应纠正措施。（详见附件“FMEA 实例”）

某家公司专门为电信供应商做客服管理软件系统，因客户对质量有要求，每次发布后都分析有哪些发版后的问题。有些软件开发相关问题不容易排查根因，例如某次发布后发现报表的数字对不上，但这些错误不是立马使用时就出现，开始的时候没问题但使用时间长了，就会产生错误。

团队花了很长时间分析整个过程，最终发现错误是由于有些内部接口，没有考虑到所有传输数据是否正常，有些有错但没拒收，导致开始时没问题，但累积多了就会出现最后的报表错误。最后也针对这问题改正了。

讨论分析如何能避免同类问题：



如果按系统产品集成的流程，识别有那些点可避免问题发生（FMEA思路）。可以在集成测试时和系统测试时，增加类似的场景，加大类似的压力测试，应可预先暴露问题；集成测试时，如果都测试所有接口传输的数据，也可以避免问题；如果我们前面做好设计和代码的评审（根源与设计有关），也应该可以避免。例如如果针对这种问题，完善评审检查单的检查项，以后应可避免同类问题再发生。

所以从这实例可以看到针对技术问题，我们可以先画出从总体流程/路线图，每一个环节有什么风险，就可以更好的帮我们挖掘根因。

质量经理：你说我们上次缺乏具体量化目标，应如何改善？

我：用水晶球预测模型可以帮团队制定缺陷范围目标：

### 团队第一轮回顾实例

- 团队分析迭代缺陷数据，缺陷是源自需求/设计/编码，得出以下分布表：

	Req	Design	Coding	
RR	1			
DR	1	2		
CR	2	3	6	
UT	2	5	10	
ST	3	9	16	
AT	1	1	2	
	10	20	34	

- 按 DRE 公式，计算出各个过程的缺陷排除率

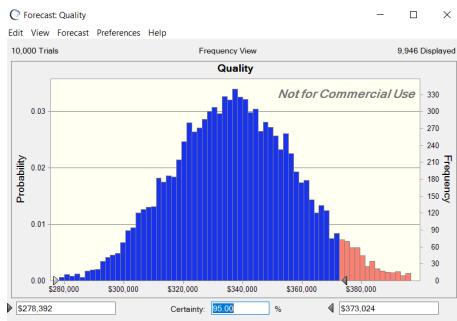
RR	$1/10$	10%
DR	$1+2/(10+20-1)$	10.3%
CR	$2+3+6/(64-4)$	18.3%
UT	$2+5+10/(64-15)$	34.7%
ST	$3+9+16/(64-32)$	87.5%
AT	$1+1+2/(64-60)$	100%

	① 修复缺陷 总工时	② 缺陷数	① / ②
RR+DR	16	4	4
CR+UT	80	28	2.9
ST	160	28	5.8
AT	80	4	20

- 从团队工时表估算各过程的缺陷返工工作量。例如最下面一行 Quality 那行输入：\$20,000 (因为验收测试缺陷平均修复工时是 20, 假定每工时成本为 \$1,000); 系统测试那行：\$5,800(因为系统测试缺陷平均修复工时是 5.8)

Zone 4 AE2:AL60		Costs	Unit Costs	Min	Most Likely	Max
Requirements	Development	\$71,898	\$844	\$7000	\$8000	\$10000
Effort	Quality					
Reqs Review		\$8,416	\$825	\$7000	\$8000	\$10000
Effort	Quality	\$15,729	\$844	\$8,400	\$8,400	\$8,400
Design		\$67,493	\$796	\$7000	\$8000	\$10000
Effort	Quality					
Design Review		\$6,308	\$764	\$7000	\$8000	\$10000
Effort	Quality	\$27,002	\$844	\$8,400	\$8,400	\$8,400
Code		\$88,301	\$844	\$7000	\$8000	\$10000
Effort	Quality					
Code Review		\$11,682	\$886	\$7000	\$8000	\$10000
Effort	Quality	\$71,958	\$847	\$8,400	\$8,400	\$8,400
Unit Test		\$144,168	\$724	\$7000	\$8000	\$10000
Effort	Quality	\$93,814	\$744	\$8,400	\$8,400	\$8,400
Integration Test		\$4	\$892	\$7000	\$8000	\$10000
Effort	Quality	\$3	\$8,924	\$8,400	\$8,400	\$8,400
System Test		\$79,004	\$881	\$7000	\$8000	\$10000
Effort	Quality	\$114,782	\$745	\$8,400	\$8,400	\$8,400
Acceptance Test		\$62,796	\$884	\$7000	\$8000	\$10000
Effort	Quality	\$101,997	\$747	\$8,400	\$8,400	\$8,400

- 估算质量总成本：水晶球会依据质每过程的缺陷分布和单位成本分布，预估质量成本的分布。按本来迭代需求引入缺陷数 =10 (3), 需求评审缺陷排除率等于 10% (4), 设计引入缺陷数 =20 (2), 设计评审排除率 =10% (4), 代码引入缺陷数 =34 (4), 代码评审排除率 =18% (1), 单元测试缺陷排除率 =35% (2), 系统测试缺陷排除率 =88% (1), 验收测试缺陷排除率 =99% (3) 来估算质量成本分布。结果如下：(括号里的数字代表在模型参数表里那一栋, 从 1 至 4。)

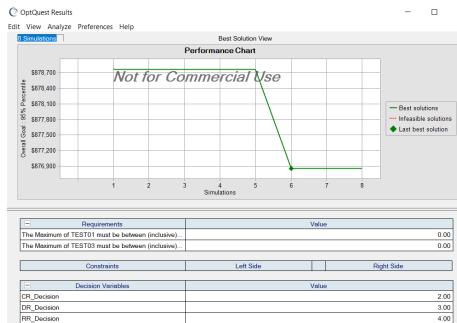


从上图看到质量成本的分布，P95 (90% percentile) = \$373,024

- 团队经过根因分析找出纠正措施包括：
  - 完善需求评审检查单，估计可以把评审缺陷排除率从 10% 提升到 80%，但因为要完善检查单评审工时也会从本来的 23 人时增加到 40 人时。
  - 使用审查方式评审后台设计，估计可以把设计评审缺陷排除率从 10% 提升到 69%，但评审工作量也会从 48 人时增加到 65 人时。
  - 使用代码走查方式，估计可以把代码评审缺陷排除率从 18% 提升到 35%，但评审工作量也会从 12 人时增加到 20 人时。

利用水晶球模型比较以上各种不同的配搭，选出总成本最低是配搭组合。在水晶球参数输入表格，在需求评审、设计评审和代码评审都加上新方法的工时和质量，在模型选最优配置需求评审可选方法（3、4）设计评审方法可选（3、4）代码评审可选方式（1、2）让模型从  $2 \times 2 \times 2 = 8$  种可选搭配中比较，挑选哪个搭配的总成本最低。

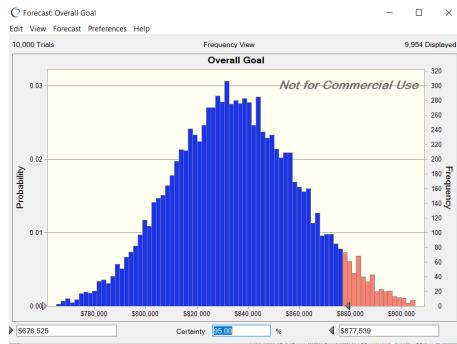
下图显示一直水晶球优化的过程，和最终的最“佳”配搭：



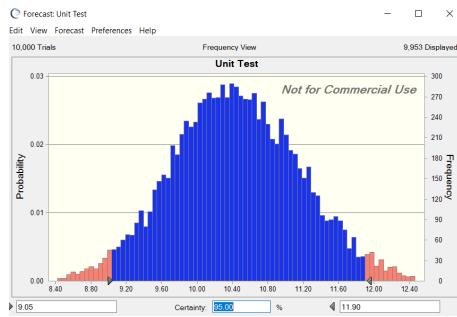
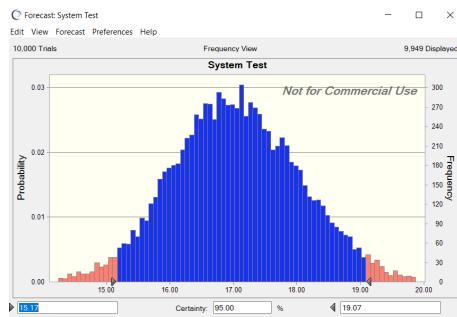
从上图看到最佳搭配是：

- 代码评审（2）采用走查方式
- 设计评审（3）采用审查方式
- 需求评审（4）不变

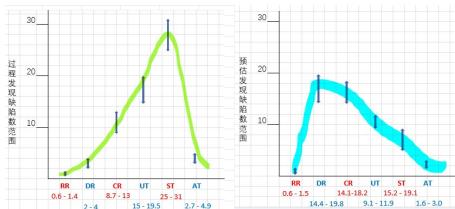
预测模型也会展示总成本的分布如下：



- 预测模型依据我们输入的缺陷参数，得出每个过程的缺陷范围，例如系统测试缺陷数 95% 上下限范围是 15.2 至 19.1。例如单元测试缺陷数 95% 上下限范围是 9.1 至 11.9



- 如果我们把优化后的缺陷分布（右图）与本来迭代的缺陷分布（左图）比较，能看出改进后，本来在系统测试才发现的缺陷大多数可以预先在评审暴露，降低返工工作量，也可以预估评审缺陷范围，如果实际评审缺陷数未能达到范围，便可预警，尽快纠正。



如果没有预测模型，我们只能主观估计会降低，有了预测模型，便可以估计缺陷分布的变化，也能帮我们挑选最佳搭配，并预估下一轮的缺陷分布范围。

#### 解读预测模型最佳搭配选择：

模型以总成本最低可以防止局部最优。例如，需求评审没有选用排除率高的新方法，因为新方法会使用较大评审工作量。如果用质量最优（质量成本最低）便必然会选用地缺陷排除率最高的方法，但用总成本来比较便可以综合考虑工作量 (Effort) 和质量 (Quality)，避免局部最优。

今次有 3 种新方法比较，共 8 个配搭。假如 3 类评审都有两种新方法可选，我们便要比较  $27 (3 \times 3 \times 3)$  配搭，每个配搭都单独模拟预估成本，然后统一比较，会非常费时，水晶球的最优功能能帮助快速选出最佳配搭。

质量经理：做了第一轮迭代后，以后的迭代有什么要注意？

我：团队累积了多轮迭代数据，便可以开始分析数据，并开始建立标杆。

#### 建立团队标杆（基线）

第一轮做模型估算，有些新方法的参数，例如需求评审排除率，都是随便预估出来。但第二轮迭代回顾时，便有实际的数据依据来调整缺陷排除率，更新模型预测参数，使下一轮的预估缺陷范围更能反应实际、更准确。当团队一直每轮都是用这种方式去完善后，就可以建立团队标杆作参考。

例如代码评审的缺陷率，从以往迭代数据可以得出上下限范围判断新一轮迭代的缺陷率是否超出范围。关于如何利用迭代的数据，用控制图判断是否稳定建立上下限范围，会在控制图（25 章）里详细说明。

注意：上面我们是说缺陷率发现的缺陷数除以迭代的规模数，而不是说缺陷数，因为每次迭代的规模都不同，迭代的缺陷数是无法比较的，但是缺陷率就可以比较了。所以利用简化功能点估算规模是团队量化管理的基础。

#### 从迭代复盘到持续改进

回顾时做好复盘根因分析，利用数据，改善下一轮开发质量。如何可以变成团队或公司的持续改进呢？

深圳有一家公司专门是做内部 IT 服务。因为要求很严格，都要经过测验验收都通过，才允许新版本正式投产（发版）。每两周会做一次发版，每次都会统计发版成功率，因为都做了很久，依据以往水平，要求发版成功率不会低于 96%，并一直都监控这个百分比趋势。某一个月发现成功率跌到接近 96%。质量改进组就开始启动根因分析。先看是哪个部门出问题引起发版率低。发现有两个部门表现最差，然后针对这两部门做根因分析找出具体原因。然后也对应一些原因做了纠正措施，比如培训评审等等，做以后发现确实那些问题解决了。发版率的水平也提升了，后面反而可以把那个变成新的基线，变成一个新的水平。到了 98%，

从以上例子看到，按成功率，或缺陷率，从趋势，制定标杆，使用根因分析，针对根因采取纠正措施，质量提升，升到一个新的水平。

### 促进公司过程改进

做好根因分析，不仅仅能帮助团队提升，也可以帮助公司加强部门之间的沟通，帮助大家改进、提升。

我本来想问某家专注医院系统产品公司的质量经理，是否有很多缺陷在后期暴露，希望把缺陷前移，减少返工。他觉得这不是他们问题的重点。

他举例，“很多需求只是简单写一句话，所以不仅仅是开发与编码的问题。我们的产品经理都没有做什么过滤分析，直接把客户的简单需求发给开发，导致后面缺陷很多，客户验收时才发现问题。”

然后他打开系统说，“例如，这条需求只是简单一句话写了客户需要简化外科医生的流程，有些审批步骤可以简化。但需求人员没有详细写清整个流程与步骤，整个流程跟相关的页面有什么特殊处理。可以想象，开发人员还是会一头雾水，做出来肯定不能满足客户要求。”

为了促进部门之间沟通，我们组织了两天互动工作坊培训，主题是：“X 通 XX 2024：提升客户体验，实现价值交付”，邀请了产品经理、项目经理、研发、设计、测试和实施/交付等都参加，分成四组，每组 8 到 10 人。

第一天早上，先用机场数据学习怎么利用 KJ 分析方法和帕累托图，让大家开始利用数据分析根因。下午，小组按自己的角色列出满意和不满意的实例，并预计一年后的理想场景，提升大家兴趣与动力，因为如果只讨论解决问题，参与者反而会没有动力。最后引入度量与分析的主要概念，如 SMART 原则，定目标。利用 GQM 概念（详见附件）收集一些能帮助分析的数据，而不是什么数据都收集。大家按希望的场景列出具体的量化目标。

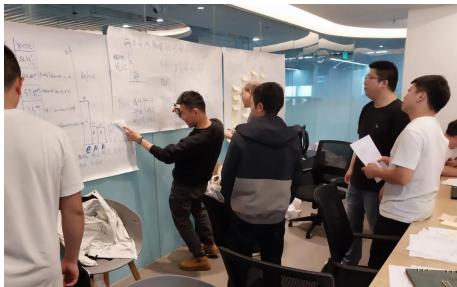
第二天，分组依据模拟缺陷数据分析根因，并利用水晶球模型预测下轮迭代效果。

我发现某些组提出的改进建议是超出团队自己可以改进的范围，便说：

过程改进不仅仅是团队提升，也需要公司级配合才有效。如何可以做到公司级提升？例如某些团队需要公司级做好新员工培训，就需要成立过程改进小组，过程改进小组，通常会按功能分工。和你们按分功能分组一样，有专门需求或产

产品经理的小组、专门做研发的小组、针对设计的小组等，每个过程领域都应该有相关的改进小组专门探索有哪些地方可以改进，如何改进。

所以我们下午就按这个思路要求每个团队写出希望改进的重点。然后让大家回顾，两轮讨论后，最终形成大家都接受并赞同的改进点和对应的改进行动：短期三个月，和长期行动计划（例如一年）。最终形成一份团体报告，向高层汇报。



虽然公司的问题不仅仅是缺陷前移，但还是可以利用缺陷前移根因分析，带出迭代回顾的重要性和量化管理，加强部门之间的交流沟通。例如：

产品经理能了解到研发人员的困难：研发代表：“今天早上有个紧急需求，下午五点钟前就要交付。”

需求人员也听得到交付人员的困难：“医院客户要求很高，但因为有些实施人员工程师经验不足，延误了整个部署，导致客户不满。”

以两天工作坊开始，再配合高层定期监控（例如改进小组每季度汇报），就可以帮公司建立持续改进的文化，加强了部门之间的正面沟通。

某项目经理参加了两天的培训后说，“虽然我们一直都有定期部门会议，但没有机会像这两天这样大家聚在一起讨论并了解，今次确实听到很多其他的部门的声音，让我们可以更全面了解到问题所在，这种沟通也能帮助大家互相协调，不会仅仅是把困难放在自己心里，但却无法解决。”

注意：在这种场景，老师（或内部教练）的任务是鼓励各人“发声”，自由发表自己的意见，反应事实，所以培训开始时要强调以下原则：

1. 让各岗位成员一起寻找改进方案
2. 看全局，大家一起寻找共同点（避免瞎子摸象）

强调从以前传统的依靠顾问专家指导思维，变为第2章提过的“所有员工参与与改进整个系统”。

所以老师不要希望在工作坊里“教”大家，使他们提升，而是大家自己通过沟通交流，发现共同点，制定改进行动。

我五年前帮另一家北京公司做工作坊互动培训时，尝试教团队如何做好根因分析，引导他们基于公司目标选择自己的度量项等。虽然最后小组还是按我的意思去写了总结报告，展示高层。但培训后果就没有下文了，所以重点不是学员学到什么新的方法，最重要让他们动起来，自己动脑筋自己发现有那些方面立马可以做改进，他们才有动力后面执行，所以应尽量减少干涉，只是要求他们按时间完成指定任务，具体什么根因，什么改进方法让他们自己想，过程中也可利用功能性分组帮助团队成长（参考附件实例）。

## 结束语

Kent BECK 先生的极限编程，快速迭代，每一轮交付对客户有价值的产出物，并得到客户反馈，而不是花精力写文档（如，设计文档）。如能做好每次迭代回顾，便可以持续改善（类似丰田方式）。

要推动公司团队做好迭代回顾，必须先获取高层的认同与支持。

为了引发领导兴趣，我先问三个问题：

1. 你们觉得占工作量最多的是哪一类工作？
2. 你们现在的缺陷绝大部分是在什么过程暴露？
3. 你们估计验收/系统测试缺陷的返工工作量是多少？

接下来，我就按“11 如何降低软件开发质量成本”讲，很多人低估了缺陷返工所耗费的工时，导致大部分的工作量都用于修复后期验收/系统测试的缺陷，软件开发的特性是缺陷越后期发现，返工工作量越高，如果每次迭代能把后面才发现的缺陷预先在前面发现并解决，不仅仅提高质量，同时因减小后面的大量返工，同时也提高团队生产率。

这方法可以有效地在头 15 分钟让管理层/开发组长体会到针对缺陷前移做好根因分析的好处。

strut

有了管理者的关注与支持，要做好迭代回顾，必须所有团队成员都参与，并赋予团队充分时间分析、讨论改进行动。

要基于数据做好回顾分析，最困难不是数据分析，而是怎么收集到真实的数据。所以必须在培训里，让学员知道为什么要收集数据，为迭代收集数据做好准备。

如果希望干系人有执行改进行动的动力，必须要他们在回顾时全心投入参与讨论，一起找根本原因和解决方案。所以培训时，不仅仅是教分析的技巧，更需要多利用互动游戏让他们可以放心发表意见。避免迭代回顾时，项目经理‘一言堂’。

当数据不是单点而是分布时，蒙特卡罗预测模型可以帮我们更好处理数据。针对如何把发现缺陷前移，模型可以从每个过程的分布预估总返工工作量成本分布，也可以比较不同的搭配，自动挑选哪个搭配的总成本最低。

分析方法和可以改进的方向很多，这部分主要以一些实例带出每一个步骤的重点。大家掌握了这些节奏之后，就可以融会贯通，持续进行根本原因分析，形成不断优化的良性循环。以提升质量为目标，不再习惯于当前的缺陷水平，觉得大量缺陷在后期测试时被发现是正常。

## 不仅仅适用于分析迭代缺陷数据

请不要误以为这部分提到的方法、工具是做好团队回顾的唯一方法。因大部分团队都有这问题，缺陷相关数据也比较容易收集到，所以建议先从缺陷前移入手，容易快速看到效果。分析缺陷排除根因取得效果后，可以继续探索其他改进方向，例如除了硬数据（缺陷数，工作量等），也可分析软数据（团员的心情，部门间的合作性等）。方法、工具会不断有新的取代，利用数据，分析根因，希望避免同类问题再发生，持续改进这些原则才是做好迭代回顾的重点。

## 驱动整个公司的过程改进

分析缺陷排除率，做好根因分析，降低后期的缺陷密度，不仅仅对团队的提升有作用，也可以帮助整个公司，加强部门之间的沟通，帮助大家改进、提升。

例如某家专门是做医院系统产品的公司。我本来想问他是否希望很多缺陷在后期暴露，希望把缺陷前移。他听完以后，觉得这不是公司的问题重点。

他举例，“很多需求只是简单写一句话，所以不仅仅是开发与编码的问题。我们的产品经理都没有做什么过滤分析，直接把客户的简单需求发给开发，导致后面缺陷很多，客户验收时才发现问题。  
然后他打开系统（他们的缺陷都在系统管理），例如，有一条需求只是简单一句话写了客户需要外科医生的一个流程的简化。有些审批步骤可以简化，但需求人员没有详细写明整个步骤如何，整个流程跟相关的页面有什么特殊处理，可以想象，这种的话扔到开发还是一头雾水，肯定做出来的不能达到客户要求。”

为了解决这个部门之间沟通，我们的组织了两天互动工作坊培训，主题是：“X 通 XX 2024：提升客户体验，实现价值交付”，邀请了产品经理组商务经理组研发设计测试，和实施交付等都参加，分成四组，每一组 8 到 10 人。

第一天，早上先用机场数据要他们用 KJ 分析方式和帕累托图开始学习，怎么利用数据分析根因。下午就是让每个小组按自己的角色列出他满意和不满意的点。并预计一年后到 2024 年底理想的状态，因为如果只是谈解决什么问题，反而参与者会没有动力。如果先采用一些希望的未来场景，他们就有动力，更积极参与。最后用一有多小时开始引入度量与分析的主要概念，如 SMART 原则，定目标。利用 GQM 概念收集一些能帮助分析的数据，而不是什么都收集。要他们按希望的场景列出具体的量化目标。

第二天，我们利用水晶球模拟数据，要求每一组用半天时间就缺陷排除率的根

因分析，并利用水晶球模型预测下轮迭代效果。过程里，组里成员就提出一些改进建议，是超出团队自己可以改进的范围。

我便趁这个机会总结给团队说：

过程改进，其实不仅仅是团队个别的提升，也需要整个公司级配合才更全面。如何可以做到公司级的提升。比如有些需要公司级做好新员工的培训等，这就需要成立过程改进小组，过程改进小组，通常会按功能分。正如你们现在分功能一样，有专门是需求或者产品经理的团队，专门做研发的团队，专门做设计等，每个过程领域都应该有相关的改进小组专门探索有哪些地方可以改进，如何改进。

所以我们下午就按这个思路要求每个团队写出希望改进的重点。然后让大家回顾，两轮讨论后，最终形成大家都接受并赞同的改进点和对应的改进行动：短期三个月，和长期行动计划（例如一年）。最终形成一份团体报告，向高层汇报。

虽然公司的问题不仅仅是要缺陷前移，但还是可以利用缺陷前移根因分析，带出迭代回顾的重要性和量化管理，大家都就可以加强部门之间的交流沟通。

\* 产品经理能了解到研发人员的困难：研发代表：“今天早上有个紧急需求，下午五点钟前就要交付。”

”需求人员也听得到交付人员的困难：“医院客户要求很高，但因为有些实施人员工程师经验不足，延误了整个部署，导致客户不满。”

以两天工作坊开始，再配合高层定期监控（例如改进小组每季度汇报），就可以帮公司建立持续改进的文化，加强了部门之间的正面沟通。

某项目经理参加了两天的培训后说，“虽然我们一直都有定期部门会议，但没有机会像这两天这样大家聚在一起讨论并了解，今次确实听到很多其他的部门的声音，让我们可以更全面了解到问题所在，这种沟通也能帮助大家互相协调，不会仅仅是把困难放在自己心里，但却无法解决。”

**注意：**在这种场景，老师，或内部教练，的任务是鼓励各人“发声”，自由发布自己的意见，反应事实，所以培训开始时要强调这原则，也可参考附件：功能性分组帮助团队成长实例。

#### 自主，减少受他人影响

- 回到现在 **Return to Now**
  - “现在想做什么？”
- 感受，并非由他人引起 **Resource your Feelings**
  - “你令我生气？” → “我自己生气”
- 取消归咎和赞赏 **Remove Praise and Blame**
  - “我伴侣拒绝我” → “我自己拒绝与伴侣关系”
- 重组“你”和“你在我心中的形象” **Recognize the difference between ‘you’ and ‘you-in-me’**
  - 我们很多时候只看到“你在我心中的形象”，并非真正的“你”

Source: John and Joyce WEIR

### 如何做好迭代回顾：“总结”

有数据才可以谈改进，有数据支撑的根因分析可以帮我们在下一个迭代做改进。因为改进花精力，所以必须有针对性，二八原则可以帮助识别应针对哪方面入手。软件开发应，软件开发团队花最多工作量的不是编码本身，用于修复缺陷工作量最多。软件开发有个特点，缺陷越早暴露，返工工作量越多，例如如果缺陷在客户使用后才暴露，可能总共要花 30 - 50 人时修正，但如果缺陷能在前面评审或单元测试发现和改正，就不会超过 1 人时。但如果回看每次开发交付，分析缺陷分布，大部分缺陷都是在系统测试或验收时才暴露，前面评审、单元测试、集成测试发现的缺陷都很少。

如果团队可以提前在评审、单元集成测试预先发现缺陷，不仅仅能提升产品的质量，让客户更满意，也可以节省团队工作量，提高生产率。

如果团队能在迭代复盘时或者迭代回顾时，一起分析缺陷排除率，针对占比最多的种类，使用便利贴方式 KJ 方式一起分析根因，制定下一轮的纠正措施，便可以把缺陷发现时间前移。

虽然很多团队的根因分析用了帕里拖图、鱼骨图的方式，这些都是基于主观判断的方式，没有真正利用基于事实的数据，而且很容易受从众心理影响，变成项目经理的一言堂，无法挖掘出真正原因，使用便利贴 KJ 方式可以帮助团队，更好从事实总结出背后的主因。团队所有成员一起，包括需求、开发、测试、项目经理等一起分析迭代的缺陷数据，可以计算迭代的缺陷注入和缺陷排除率，利用蒙地卡罗预测模型就可以把本来只是定性的一个改进目标变成定量。

例如，如果需求评审的缺陷数未达到本来预计的范围，就可以预警团队可能评审力度还不够，没有得到效果，很可能不少缺陷，还是会等到后期才暴露，就可以立马加强力度，提高本迭代质量提升的概率。

根因分析不仅仅能改善团队产品质量问题，也可以用于其他改进，如用于两天的工作坊互动培训中，加强公司功能部门之间的协调，配合公司发展目标，相关过程改进小组制定短期和长期改进量化目标，帮公司过程改进升一台阶。

### 乔布斯，当 NEXT CEO 时，被访问，谈质量：

整个质量提升的道理其实很简单，是一个重复的过程，然后我们需要不断去看，有哪些无效的环节要省略，哪部分要重新设计，不断试验、提升，就这么简单。重点是所有的提升都应该是科学化的，有数据而不是泛泛而谈。

极限编程 (XP) 的最佳实践可帮我们发现现在的开发过程有那些不足，也有助找出具体根因和纠正措施，下一部分会探索里面的实践能如何帮助团队提升软件开发质量。

## 附件

### FMEA 实例

(FMEA: Failure Mode Effects Analysis)

大家可能都遭遇过由于没有管理好时间，而导致迟到的情况。以坐飞机为例，从

离开酒店到登上飞机的过程中，很可能因为一些风险导致最后没搭上飞机。你出发的时候，可能用不同的交通方式：出租车、机场大巴等。如果你不能在起飞前 45 分钟到达机场办理登机牌，你便搭不上飞机。但是你拿了登机牌也有可能最后搭不上，因为飞机都严格执行起飞前 15 分钟关舱门所以你拿了登机牌，还要在起飞前 15 分钟到达登机口。

Fig 1 登机过程



要赶上飞机其实是个过程，中间有很多环节会导致最后失败，所以我们可以通过 FMEA 过程分析来看如何减少失败的概率。

Fig 2 FMEA 例子

Process / Product Failure Mode and Effect Analysis											
Process or Project Name	1	Process Responsibility	1	Process ID	1	FMEA Number	1	Plan Date	1	Prepared By	1
Business Year (YY/MM/DD)		Key Personnel					<th> </th> <td><th> </th><th></th></td>		<th> </th> <th></th>		
Core Team							<th> </th> <td><th> </th><th></th></td>		<th> </th> <th></th>		
Process Step	Potential Failure Mode	Potential Effect of Failure	S	Potential Causes	O	Current Process Controls	R	A	Recommended Actions	S & Target Completion Date	D
2 出发	4 不到45分钟到达机场	5 无法登机	6 飞机晚点	7 误机	8 10	9 11	10 12	11 13	12 13	14 15	16
酒店去机场	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机
过安检	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机
去登机口	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机
拿登机牌	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机
登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机	无法登机

Fig 3 打分参考

Severity 严重程度	Occurrence 发生频率	Detection 容不容易检测	Risk	
			High	Low
10 危险没有警告	非常高的几乎不可避免	不能检测到...或者是非常低概率的检测		
失去主要功能	很高的反复失败	远程或低...检测的机会		
失去次要功能 小缺陷	中等的失败 偶尔的失败	低检测概率 中等检测概率		
1 几影响	极低	必然的概率		

以第一个失效为例：如发生便坐不上飞机，所以严重性是最高 10。发生的概率还是比较高 6。是否容易预防、预警。因不熟悉当地情况，加上我主要靠问酒店前台，有时候她也不清楚，所以我定 6。RPN = 10x6x6=360。

从以上登机的例子可以看出 FMEA 是以整个过程来管理风险，比如在出发前，就要查询一下各个交通工具要花费的时间，比如你发现在南京，从酒店坐地铁要转车，要花 1 个小时以上，如果时间太紧就来不及，宁可多花钱打车，才能控制风险。当你拿了登机牌，还是要经过安检，还要从安检走过登机口。有时候机场很大，到达登机口也要花很长时间，就要先问好路径，提前计算好时间，才不会误点。

过了安检到登机口要 10 分钟以上，就要在起飞前的 35 分钟完成通过安检，才来得及。这些都可以通过 FMEA 的形式把整个坐飞机的过程识别出来，找各个阶段会出现的问题，就知道如何控制。

以这坐飞机的风险为例，我自己就多次未赶上飞机，原因很多，但总结一下都是习惯没改过来。我应依据以往差点赶不上的经验，回顾一下，确保每一个过程都控制住，就不会后面再次出问题，这个和企业做风险管理概念一样。有度量才有管理。人和公司一样，很多做的事情好像是自己主动去想，其实很多都是潜意识习惯，如果你没有定一些量化的控制手段，就不会提高这方面风险意识，还是会搭不上飞机。我先回顾以往几次搭飞机的情况，并把每次到达机场的时间和到达闸口的时间写下（详见下面 Fig 4）

Fig 4 上面是机场柜台关闭（45 分钟）前到达时间，下面是关闭口前到达时间（分钟）（X 代表赶不上飞机）

航班	NJ-SJZ	HKG-CD	CD-SJZ	SJZ-CD	CD-BJ
到机场	x	12	10	6	10
到闸口		1	2	1	x

最近一次赶不上飞机是因为未能在关闭门前到，之前已发生过两次 --- 我刚到闸口，就到时间马上关闸了！

如果我把经验教训记下来，下次做好时间管理，就会避免后面的延误：有了数据，我们便可以更有效在回顾时做好根因分析 (CAR)。

以这登记延误风险为例，可以使用 FMEA 分析每一失效点，例如过安检（因没有预留足够时间）与从安检到闸口太长时间等的发生概率都很高（前者 8，后者 6）

为了避免问题再发生，就要定一些具体的计划，最终希望把误点减到零。

在登机这个环节，可利用什么有效方法/工具，帮助改善？

- 每次出发去机场前都查询各种交通方式的时间与风险（概率），预留足够时间，降低失效发生的概率
- 拿了登记牌后，都计划好必须起飞前 35 分通过安检，45 分前开始安检

在多次没登上飞机后，我发现平常的手表没有正负 5 分钟的概念，但是如果用电子手表，对时间的感觉可以准确到了正负 1 分钟，就能够更好把控时间。

Fig 5 平常用于培训 / 评估计时的电子钟



经过这次误机，我就买了个电子手表，取代传统针式手表，希望对日后的不迟到有帮助。

效果：这故事发生在 2019.9，后面我按这些计划，一直都没有再出现赶不上飞机的情况，我每次都提醒自己必须在起飞前 60 ~ 90 分钟到机场，30 ~ 45 分钟前到闸口。分析的目的是提高个人风险意识，避免问题发生。

## 功能性分组帮助团队成长实例

系统，包括团队、公司，的成长都依靠不断集合内部的差异，如果完全不能接受不同意见，就无法进步。基于心理学家 Yvonne Agazarian 系统中心理论 (Systems Centered Theory)，团队必须先从成见分组 (Stereotype subgroup) 变为功能性分组 (Functional Subgroup)，才能成长。

首先，要让大家觉得有共同点。基于这基础才可以听从不同意见。但首先要让大家愿意接受不同意见，不然的话就会变成争吵，不会有效果。有人提出不同意见后，必须有人附和，成为他的战友，才能形成新的小组。有了新小组以后，团队需要把不同小组的立场合并起来，变成一个新方向、新思路，整个团队就有成长、进步。

### 场景实例：

团队讨论如何完善社区里面的教育，大家讨论个人的希望。

A: 好像我们大家的想法都很类似。

内部教练：可否举个例子？

B: 我们都想为自己和我们后代有个终身不断学习的机会。

C: 好像我们都没人提过大学，不知道为什么。

D: 但很多人负担不起大学学费。

E: 我觉得每个人只要想受到教育，都能做到，我就是靠自身努力。最后完成大学教育。

F: (开始跟 D 形成小组) 是的，但教育确实越来越昂贵，我估计我们当中不少人负担不起那些美国东岸名校的昂贵学费。

G: (有成见，觉得那些不能完成大学教育的都是没有动力，没有上进心) 我还是觉得任何人，只要有动力都能做到。

如果没有人附和 E 的观点，就可能变成他的个人独角戏。后面有两个可能的发展场景：

场景一 C 附和了 E 的观点。

C: 我的经历确实和 E 说的类似，辛辛苦苦最终完成了大学教育，但确实很艰苦。如果我当时没有舅舅的支持，肯定完成不了。

场景二没有人附和 E 的说法。

内部教练：有没有其他人也有经过自己努力完成大学教育的经历？

(注意：内部教练千万不能说：“有没有人赞同，人只要有动力都能进大学？”因为这只是个人观点，并非事实)

A: 我确实也完成了大学教育了，但也经历了很多困难。

G: 我读了一年，因为再得不到奖学金，没有办法，只能退学。

到了这个点，整个分组就比较复杂，虽然生成了新小组，E 得到一些回应，但也有些人提出不同的经验，需要有说法把新小组归纳。

接下来，

C: 我看来这个很简单，有些人能完成大学教育。有些人虽然很希望完成，最终还是完成不了，但不是所有人都想或者必须完成大学教育。所以我们的改进任务应该是帮助所有人学到他希望学习的东西。

有了以上 C 的说法，整个团队就可以进一步成长。

## GQM

- G:Goal (目标)  
 Q:Question (问题)  
 M:Metric (度量)

收集数据很花精力，所以必须只收集与改进目标相关的数据，但如何能联想到那些度量使相关？GQM 利用问题让我们找到相关的度量。

### 1. 制定目标

<b>描述类</b>	通过分析、辨别，《改正问题的时长、客户发现缺陷的源头、项目延误的原因等》 希望《缩短解决客户问题的反应时间、降低项目开发成本、了解...、管理...等》。
<b>分析与了解</b>	从（开发团队、客户、管理者）的角度分析（成本、缺陷、变更、有效性等）。

### 2. 针对目标提出问题

<b>通过降低分析改正问题的时长，缩短解决客户问题的反应时间。</b>
<input type="checkbox"/> 改正是否经过评审和测试? <input type="checkbox"/> 估计改正问题的时长是否正确? <input type="checkbox"/> 发现问题的平均所需时长? <input type="checkbox"/> 解决问题的平均所需时长? <input type="checkbox"/> 需要返工的客户问题占所有客户问题的百分比?
<b>管理层通常会问 ...</b>
<input type="checkbox"/> 我们能否识别? <input type="checkbox"/> 需要多长时间? <input type="checkbox"/> 需要哪些人...、多少人? <input type="checkbox"/> 成本多少? <input type="checkbox"/> 如何衡量改进?

### 3. 针对问题选择度量项

<b>目标</b>	从项目A开发团队的角度分析交付产品的软件复用有效性。
<b>问题</b>	
<b>重新开发的模块占比？</b>	M1: 针对每个模块，是否重新开发（是、否）?
<b>问题</b>	
<b>复用模块占比？</b>	M1: 针对每个模块，是否重新开发（是、否）?
<b>问题</b>	
<b>模块复用与可靠性有什么关系？</b>	M1: 模块复用占比（所有模块）? M2: 针对问题数多的模块，复用占比? M3: 针对问题数少的模块，复用占比?

### 4. 如何收集数据

- 谁来收集
- 什么时候收集？
- 如何能有效收集数据并也保证正确？
- 谁是数据分析的对象？

### 5. 收集、确认和分析数据，并反馈，让项目组制定纠正措施

### 6. 改进后分析数据，判断达标的程度

### 7. 做反馈，让所有利益相关者讨论

有了目标与度量项，我们便可以制定度量计划。

## 参考 References

1. WEISBORD, Marvin: FUTURE SEARCH: Getting the whole system in the Room for Vision, Commitment, and Action (2010 3/e)



## **Part VI**

### **代码质量改进的基础**



这部分参照 Jeffery 先生总结极限编程 (XP) 的三层图 (在第一章里)，利用实例，先从内层的测试驱动开发 (TDD)，重构 (Refactoring)，与结对编程 (PairProgramming) 等核心最佳实践开始，然后到中层的持续集成 (Continuous-Integration)，最后是外层的团队各角色协作/互补 (Whole Team) 和如何做好需求分析等。



# Chapter 14

## 测试驱动开发

### 测试驱动开发

开发人员问：为什么要这么做？写完程序后，然后再写测试用例，不是更正常吗？

### 为什么我们要测试驱动开发

精益是敏捷中很核心的概念。最近发生的一件事帮我回答了以上问题，也让我体会到 TDD 与精益的概念是如何协同的。

我有一位拥有三十多年电机工程经验的老同学李先生，他在对 Linux 的研究方面一直都很有兴趣，这几年又开始玩树莓派，目前他对此非常精通。

最近我找到他，希望在树莓派上安装 Ipython, Jupyter Notebook 等开源应用软件。一直以来，李先生的理念就是尽量用最简单、最轻易的方法去解决客户的问题，而不是浪费资源。

例如，95 年开始尝试建立 wiki 服务器给客户使用的时候，他就建议我用树莓派来建：

他说：“树莓派不仅省电而且快，也不知道你这个概念是否持久，我不会浪费时间——在大型服务器上安装 Linux + Mediawiki，如果你觉得这（树莓派）可以，便继续用。如果觉得它性能不够了，下一步我帮你换更大型的服务器。”



一到现在，已经 7 年了，我们一直还是继续用树莓派来协助评估（树莓派从原来的 Pi2(1G 内存)，现在已经是 P4(4G 内存)，已经服务超过百家客户了，这证明他的思路是对的。

回到我刚才说要安装 Jupyter Notebook，他安装好那个程序包后，一直没动。我问他什么原因？他说：“很简单，我不熟悉软件工程，也不懂如何测试。肯定要找你测试一下，我才知道有没有做错。这样可以避免无谓的返工。”

所以我们就约好周日我回到办公室跟他远程测试。

李先生打开树莓派后，我用 VNC 客户端进入他的树莓派，打一些最基本的 Ipython 指令，验证没问题。

下一步我要跑一些比较复杂的、要用到 Pandas 包的 Python 程序，却发现还没有安装 Pandas 包。他马上到网上去找安装介质以及树莓派如何安装 PANDAS 的方法。处理好后，我再跑对应的程序，都跑通了，包括之前跑过的指令。前后两个小时，按本来计划，测试成功。

我回想一下，其实他的思路就是精益。用有限资源、以最低投入，达到客户要求，不多做。每当李先生做完一步，如果没有通过测试，就不会浪费时间走下一步——这也是精益和测试驱动开发的原理。

### 每小步验证

工程类的本科生都要在最后一年做毕业项目。当时，1983 年，导师建议我们尝试参考一些常用的语音发声算法，在专门做信号处理的芯片上写程序，读出一些英文字或句子（与现在不同，当时这项技术还没有成熟，很多大学还在研究），虽然我预计有不小的技术难度，但觉得这很先进，很感兴趣，便与另一位同学合作，开始制定项目计划。

我们很努力，全情投入，一面要研究语言发声的算法，一面还要并行设计电子线路和软件等。我们用了 2~3 个月的时间做了整个电子线路板的硬件，同时还设计了整个软件架构，并使用计算机模拟，验证每个字实现算法后的发声效果，因为最后一年课程很多，时间过得很快，从 9 月开始准备一直到次年 3 月，软硬件的设计与开发终于都完成了，但是不知道什么原因就是没有声音，更不要说能读出一些字和句子了，最后项目以失败告终。

在这之前的一年，我有幸在大学的第三年进入大东电报局 (Cable & Wireless) 实习（当时香港的所有国际通讯都是经过大东），实习的部门（负责电报业务运

维) 正如火如荼开发一套新的电脑系统以取代本来基于 UNIVAC 的电报系统, 总工程师让我用半年时间, 在一个微机上编程, 做一个系统, 以从那些电脑上收集重要信息, 如果发现异常就警报。头三个月都是花精力做整个系统设计, 也买了一些展示电子版, 准备用来展示, 但由于经验不足, 半年后最终什么都展示不出来。

到了 2000 年, 在我兼读软件工程硕士课程时, 开始接触到敏捷开发, 才了解到两次项目失败的主因: 不应该花大量时间去做前期设计——希望有一个完美的设计, 而是应该一步一步迭代, 先做一些最基本的简单功能, 逐步优化。例如, 在我的毕业项目中, 应该先做出最基本的硬件、软件, 起码能够发出声音, 因为没有前人做过, 整个项目是从未做过的实验。

敏捷大师 Dave Thomas 先生在 2015 演讲里提出敏捷软件开发的核心是:

1. 向你的目标迈出一小步。
2. 从反馈调整你的理解。
3. 重复。
  - 当两种或以上选择的价值大致相同时, 选一条让未来更容易修改(软件)的路径。

这些经历让我体会到为什么我们在写程序时, 必须先想一下该怎么测试。

## 用 TDD 开发程序

程序员问: 为什么我们要针对这么小的部分都做单元测试?

答:

1. 小的单元测试容易写, 比较简单, 不会花费很多时间。
2. 因为任何程序都会按时间变化, 越来越复杂。有了单元测试, 后面的测试就有依据, 知道改动是否出错。没有单元测试的话, 就没有信心去改程序, 不知道是否没问题。

(这和我们工程的概念一样, 把工程细分, 每个子系统都要确保质量达标, 尽量找出缺陷, 才进行下一步。)

如不相信请看附件“从 MIT OCW 学写程序”, 我从头学编程的实例。如果老师没有提供单元测试和相关数据, 学生写完代码后是无法验证代码是否通过, 是编程的基本要求。为什么专业程序员的程序反而可以不需要先通过单元测试, 写完直接便交给测试人员做系统测试?

网上有关于是否要 TDD 的争论:

- 一派是坚持要先写测试用例, 再写编码。
- 另一派觉得 TDD 太多余, 如果功能测试做得全, 也不一定要有单元测试。

这个争论让我想起以前教 ACP 敏捷管理的一个原则, 叫“守破离”。

### 守破离

英文翻译成“Shu – Ha – Ri”，意思是学习合气道，要先从基本功出发（守），按规则一步一步做。当你理解以后，就能融会贯通（破），融合管理到一定境界就是大师级了（离）。

像宫本武藏（江户时代著名剑术家）一样，一生赢了六十多场决斗，然而到了晚年，他在《五轮书》中总结，不要局限于某种刀法，最重要是抓住原理。

敏捷大师 Mr Cockburn 有一次到企业做 Class-responsibility-collaboration (CRC) card 咨询培训，因为他很熟悉 CRC 方法，他觉得只要有助于面向对象设计，不一定要按原本的方式，可以简化。但学生都不熟悉，一直在问“请你告诉我们从头到尾，每一步如何做，我们照着做”大师没办法，最终按最原始的方法，一步步来教如何去做，学生才能懂，这是守破离的“守”。

TDD 测试驱动开发也一样，把 TDD 看成“守”，当没有概念的时候，还是要从最基本的概念入手，我们每个程序都应该有测试，所以自然的顺序是先写测试用例，再写代码，这也能更好地帮助你理解程序的功能。

所以 TDD 就是做好程序基本功的基础，当你已经掌握这些基础了，就可以灵活处理，可能不一定要每次先写测试用例，但底线是每个代码都要有单元测试。

但是有些人觉得，只要有功能测试，单元测试可以不做。

功能测试是黑盒测试，从客户的角度测试总体功能，无法真正判断代码是否正确。反过来，单元测试是白盒测试，从代码的功能角度，验证代码是否正确，所以功能测试是不能替代单元测试的。尤其是当代码有功能上的变化时，就更需要有单元测试来验证这些改动是否有误，所以单元测试的复用率非常高。

大家正在越来越多地使用自动集成工具，如果有单元测试，那么每天的自动集成不仅仅仅是看编译是否通过，还需要通过单元测试，才能更好地保证代码质量，这就是很有效率的回归测试。（回归测试：为避免因代码修改，导致原先通过的测试不通过，所以之前通过的测试用例也要重复再测。如果手工测试，为了节省测试时间，只重跑部分重要测试，但如果自动化，就可以轻松地重跑所有测试用例。）

### 单元测试与 TDD 的好处

上周在杭州跟一位资深的开发主管对话，他回国前在日本工作了接近 10 年，他说很佩服日本注重开发质量，例如很注重单元测试，所以回国后在团队严格要求要有单元测试。

我问：现在有很多静态代码检测，为何还要单元测试。

他答：目的不同，静态代码检测只是用过去的历史经验去检查常见的语法、命名问题，但如果是设计本身的逻辑问题就无法查出。以他多年的经验，必须通过单元测试才有信心确认这个程序是否正常，单靠集成测试、功能测试是无法确保程序的每一部分都是正常的。所以他严格要求团队，代码经过检验后（自动静态检验或代码走查），还必须做单元测试才能进入下一步的集成、系统测试。

单是靠集成测试、功能测试是无法确保程序每一部分都正常。

他还说 TDD 有三点好处：

1. 让开发人员在编码前去理解设计和需求。
2. 让开发人员知道代码可验证性的重要。
3. 强迫开发人员主动与设计和需求人员沟通，否则无法设计出单元测试用例，但 TDD 对团队成员素质要求较高。

## 结束语

开发人员用 TDD 编程，体现每走一步前必须验证通过，减少返工，精益思路，是中初级程序员健康成长之路，养成好习惯。

但若你是九段高手，因已经过了以上基本训练阶段，可能便不再需要，好比合气道“守破离”成长路径。

有些敏捷大师，已经累积有 20 年以上编程经验，例如上面提到的 Thomas 先生，他已经很少写单元测试，因为他觉得已经不需要依赖单元测试来验证代码是否正确。

## 附件

### 从 MIT OCW 学写程序

第四天早上 10:00 我终于把所有功能写完并通过自动单元测试！  
(趁十一长假做编码练习题。上次做题已经是春节后，在集中隔离期间做过两题。  
这次的练习题只有十个功能，本来预计可两天完成。)

练习题 (Problem Set)——分析美国过去温度变化，提供了美国超过 20 个城市的每天平均温度。(从 1961 年到 2015 年)

- 先写基本功能：
  - 画散点图，回归分析，计算标准差与决定系数 ( $R^2$  Coef. of determination)
- 分析过去的五六十年数据，来判断美国或全球是否在暖化  
给学生的文件包：
  1. ProblemSet5.pdf：分成 A、B、C、D、E 部分，详细说明每部分要开发的功能
  2. PS5.py 程序模板，学员填入代码
  3. PS5\_test.py 自动单元测试模块，自动测写好的 PS5.py

回顾一下，像我这种 Python 新手，因为不熟悉 Python 语言的属性与特性，必须按部就班，一步一步来写，欲速则不达。

更重要是体会到个人如何记录数据并分析：

### 记录时间

十多年前我在美国考 CMMI 培训师，被观察时，观察老师会在培训后告诉我，每一个模块用了多少时间，与计划对比。从此以后每次培训我都会习惯记录每个模块所花的时间。

### 怎样做

培训时，我都会在桌上放一数字电子钟，记录每一个模块的开始时间与结束时间。上完一天课，晚上在电子表单计划时间旁边写上每个模块的实际时间。这三天半，我也是用这种方式记录每个模块的时间。

### 记录缺陷

因为每个功能都很小，不到 20 行，所以没有正式把缺陷与返工工作量记下来。尤其是一些小的语法错误问题立马就改正了。但影响较大的重大缺陷，我会记在模块旁边，算是这个模块花的时间。

个人记录实际时间没有想象中这么困难，只要每天晚上简单按当天本子的数据更新电子表单，避免遗忘。

开发功能	开始	结束	耗时	缺陷	LOC	修改	备注
1) generate_models	10:30	12:00	3		7		
	13:00	14:30					
2) r_squared	14:35	16:41	2	数据类型	13		
3) evaluate_models_on_training	19:15	21:20	3	前后顺序	13		
	9:45	11:00		不熟悉图表规则			
4) Problem 4.I			1		12		
5) Problem 4.II			1		10		
6) gen_cities_avg	11:00	12:30	4	数据类型	15		
	16:38	17:54					
	19:36	20:34					
7) moving_average	17:00	18:00	2	读取语言 参数值	14		
	20:10	21:10					
8) rmse			1	复用，有些变量没 有改好	7	2 复用 2)	
9) gen_std_devs	9:45	13:30	10	错误理解需求	15		
	14:30	17:00					
	8:25	10:00					
10) evaluate_models_on_testing			1		13	1 复用 3)	
					28		

### 回顾分析

整个编程结束后，写上每一个模块的实际代码行数。注意：如果模块是复用其他模块代码，便需要注明改动的代码行数。例如：evaluate\_models\_on\_testing13 行只修改了 1 行。

分析：除了标准差模块（gen\_std\_devs）以外，其他功能都能在 1-3 个小时之内完成。

计算标准差的功能花了 10 小时，问题出在我编码时没有详细看需求。开始以为是求每个城市的温度标准差。然后再求标准差的平均数。

测试不通过后，也没有再看需求便假定求当年几个城市所有当年每天温度的标准差，还是不对。到了第四天早上再看看 ProblemSet5.pdf 需求描述才知道我一直误解了这功能需求。

gen\_std\_devs 对应每一输入年份，返回一个数值，依据以下计算：

- 1) 按输入的所有城市，计算当年每一天的平均温度（所有城市的平均）
- 2) 计算当年每一天平均温度的标准差

所以如按我本来的算法，算一个城市是正确，但两个或者多个城市的时候，答案就比正确答案大。

针对这个计算标准差问题，一直以为是语法问题公式计算错误问题。但我用一些简单的数据检验去，未能发现任何不正常。来来去去都没有实际的进展，差点想放弃。

#### 经验教训

- 最佳实践：尽量用最小的行数完成功能
  - 像 Python 这类成熟的语言，绝大部分的功能都已经具备
  - 尽量避免基础方法编写一个已有的功能 (reinvent the wheel)，除了耗费时间外，还会写多错多
- 写任何程序前必须先有自动化测试用例
  - 虽然自己可以先想一些数据自己简单测一下
  - 但替代不了另外一个人写的测试用例。跑通才有信心程序正确（所以自动单元测试是使用最多的程序）
- 架构与设计很重要
  - 校方除了提供测试脚本外，还提供了写程序的模板 (PS5.py)
  - 每个功能都明确，输入是什么，什么数据类型，输出是什么
  - 学员自己只需要填写实现的代码部分
- 先把握好新功能的用法与结果
  - 前面发现不少缺陷，是因为不清楚某功能的用法
  - 在写代码之前，自己应开个沙盘，用小程序先试验一下这计算功能有什么参数，会出什么结果
- 先评审后测试
  - 在跑程序之前，屏幕展示或打印出刚写好的代码，用铅笔在白纸上按程序的逻辑，从头思考一下每个数据的种类和中间的变化
  - 盲目地去跑程序或测试，最终只得出测试失败，但还不清楚错在哪里，如何修改。所以不如在跑程序或测试前，自己先动脑筋模拟一次，仔细想通每个数据的变化



# Chapter 15

## 代码重构

重构代码的主要目的是使代码更容易维护，其他人更容易读懂。SOLID 原则是做好面向对象设计的基石，例如随便搜索下面 SOLID 表格的关键字，可以找到很多参考资料。(SOLID 是取下面 5 原则的首英文字母)

SRP 单一职责原则	Single Responsibility Principle
OCP 开放封闭原则	Open Closed Principle
LSP 里氏替换原则	Liskov Substitution Principle
ISP 接口分离原则	Interface Segregation Principle
DIP 依赖倒置原则	Dependency Inversion Principle

也有很多参考书，如 Gamma 和 Fowler 的经典书，例如”Design Patterns: Elements of reusable Object-Oriented Software”里面有对每一种设计模式的介绍与详细例子。

### 如何学好 SOLID，打好基础做重构

培训经验：以前我们传统教学方式做重构设计模式培训，分别解释、举例说明二十几个设计模式，学生听完后，很快就忘记了，没有任何效果。  
要有效学习，程序员必须积极动手，不能单靠理论课。从 Weinberg 先生教程序员的例子（详见附件）看到，如果先在讲课时强调某些重点，然后配上编程练习，学员才能真正吸收，后面能在项目中用上。

## 如何提升学员的兴趣与动力

我们后面改用工作坊的培训方式，减少理论讲解，要求学生在电脑写小程序，效果比本来仅仅是讲的好。有些领悟能力比较强的学生，都可以很好完成一系列的练习题，但还是有 40%-50% 的人跟不上。我们在想问题在哪里？如何可以改善？

跟得上的学生本身领会力就比较强，软件开发的基础也较好，所以可以很容易跟得上我们设置的编程练习，在课堂上就可以按我们的设计完成大部分的编程练习。问题出在那些比较弱的学员，本身对编程和面向对象一知半解，开始时候觉得写小程序跟不上那些其他高水平的同学，后面就放弃了。

培训新入职员工也面临类似问题，而且如果未能在头几个月把重点学好，养成习惯，后面便难以改正。

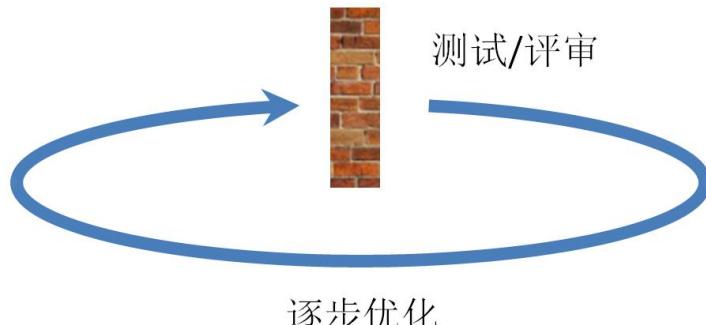
七八十年代，麻省理工科学家 Parpert 先生，他本来一直研究人工智能，他发现很多美国小学生都跟不上学校的数学与几何课，根因是老师用传统教学方法，学生没有兴趣学，他后面就利用乌龟几何 (Mindstorm) 做实验，让小学生可以轻轻松松用电脑编程、玩游戏学习几何。（详见附件）

所以后面我们就利用乌龟学数学的思路，设计了一系列的基于设计模式的练习题。设计模式最大的好处就是方便后面需求的更新。所以我们把那些常用的设计模式写成模块，就像乐高积木一样，让学员可以调用，然后学生就可以像乐高积木，尽量利用那些设计模式的组件，针对课题目设计程序。

做了完了第一段后，就有些改变：新增或者减少需求，教他们怎么可以再利用设计模式更新程序，达到题目的要求。我发现利用这种方式，学生后面在实际工作中就更能用上设计模式，因为与之前单纯对应题目编程不同，学员有更大空间设计总体的架构与功能，与前面提的美国小学生一样，因有兴趣，便能很快学到面向对象的编码原理。

## 什么时候做重构

写程序与写文章类似，都是一个逐步优化的过程，初稿可能先写一些重点，里面的细节需要不断的评审、优化，就更不用说语法、错别字了。



虽然现在我写程序的时间不多，但还是依据同样思路。先把程序写出来，基于测试驱动开发的原则，通过单元测试，但通常第一版程序有很多不足，比如不能达到 SOLID 等面向对象要求，所以会依据设计模式原则优化代码，并利用自动化

单元测确保每次改动后程序可以跑通，所以写代码也是逐步优化的过程，只是代码错了一点就会跑不通，文章因为容错性强，错个别字不影响读者读懂。专业软件工程师不会等到最后关头才去重构，而是持续的小重构，一直考虑软件的维护性、扩展性，如果达不到质量水平，宁可不发布，好比具备专业手艺的陶瓷工匠会砸碎不达标的陶器，或作曲家宁愿烧毁自己不满意的作品而不发布一样。

## 代码规范

制定编码标准是其中一条 XP 实践，因为若要写好程序、减少错误，必须预先把常见问题罗列出来，提醒大家。

客户：我们一直没有，可否直接参考其他公司的标准？

我：某公司想快速通过认证，但公司一直都没有自己的标准过程，顾问说：“不担心，我们可以提供其他已通过认证公司的过程”。你觉得公司最终会有质量改善吗？

客户：应该没有，因过程不是从本身实战经验出来，对团队提升质量没有用。

我：非常赞同，所以编码标准也必须归纳自己动手的实战经验，像刚才乌龟几何一样，让学生自己经历过解决问题，有及时的反馈，他才能真正学会。编码规范很重要，但规范不应从人家那里复制过来，而是从自己常见的问题总结出来，对团队才有意义。

## 持续集成

文章、书本等文字性的不像代码、软件，错一个字都会导致跑不通的问题。所以几十年前人们无法想象现在可以做到这么大型的软件系统开发都起码要几个月，系统规模也不会太大。现代软件开发有很多辅助工具，可以一边写一边检测，也有各个层次的测试来确保软件没问题，并能自动化，让团队可用每天发布，也能支持大型分布式系统。下一章节，我们看看如何利用自动化工具帮我们做到持续集成、持续发布。

## 附件

### 小孩利用乌龟玩几何游戏 (Turtle Geometry) 的例子

美国传统怎么教小学生学数学、几何。传统是用纸和笔的方法，小学生就没有兴趣学，导致跟不上。原因很简单，我们以数学为例，现在很多小学生学数学，还是我们几十年前那种，背九宫表算乘除。以前我们的年代没有计算机，电脑就更不用说了，所以逼着人要背那些表，就跟古代学习用算盘计算一样。但现在计算机已经可以随时买到，且很便宜，手机也有计算功能，所以他们就认为这种传统的学习数学方式已经无用，几何也一样。如果学生比较擅长数学还好，但假如有跟不上的，就觉得会觉得自己不是学数学的这块料，后面就放弃了数学。于是 Parpert 先生教小孩利用乌龟玩几何游戏 (Turtle Geometry)：



学生可以简单利用往前往右往左的命令，画出正方形、三角形：

```
TO SQUARE
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
END
```

```
TO SQUARE
REPEAT 4
    FORWARD 100
    RIGHT 90
END
```

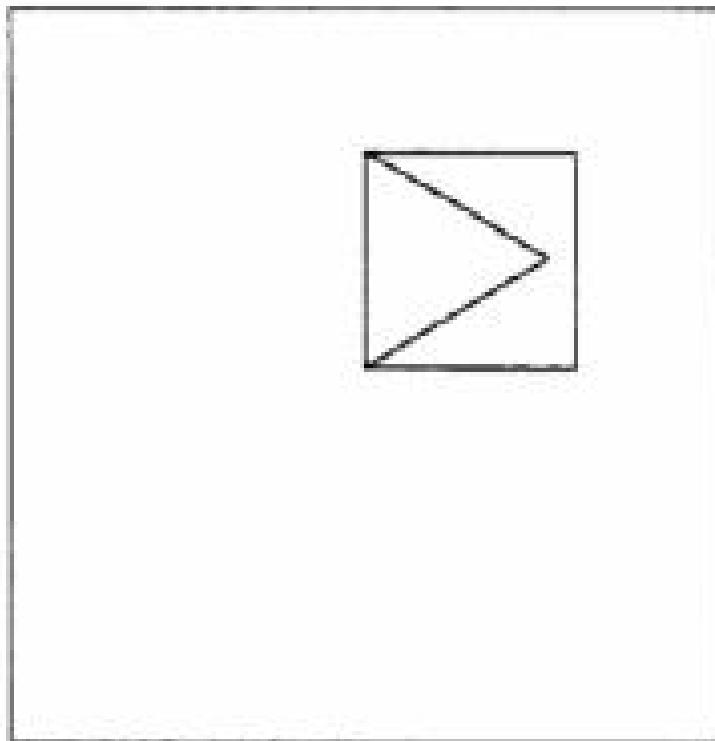
```
TO SQUARE :SIZE
REPEAT 4
    FORWARD :SIZE
    RIGHT 90
END
```

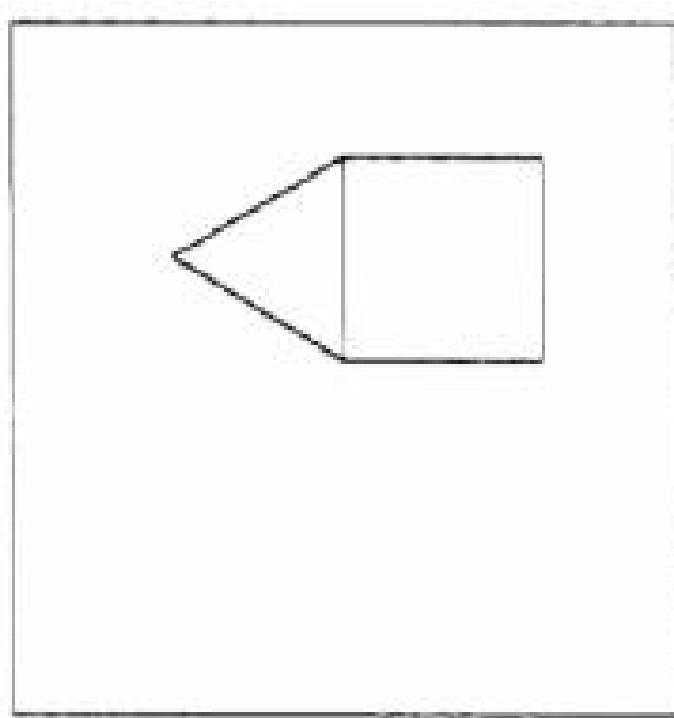
```
TO TRIANGLE
  FORWARD 100
  RIGHT 120
  FORWARD 100
  RIGHT 120
  FORWARD 100
END
```

```
TO TRIANGLE :SIDE
  REPEAT 3
    FORWARD :SIDE
    RIGHT 120
  END
```

编码有误，学生自己调代码，把三角形的方向改正，才能画成一间屋：

TO HOUSE  
SQUARE  
TRIANGLE  
END

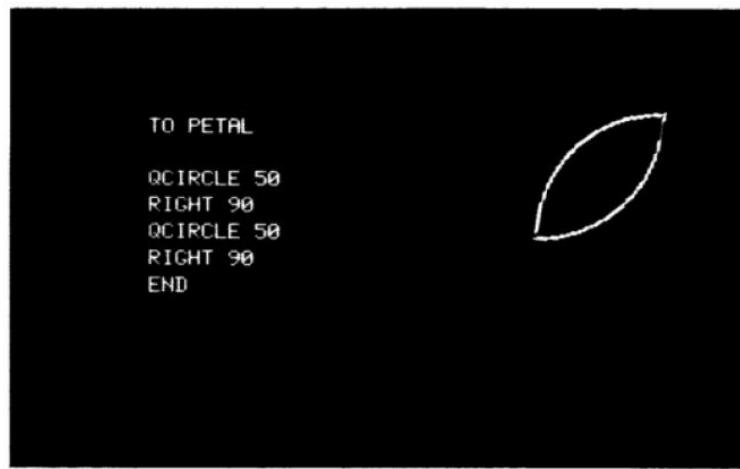




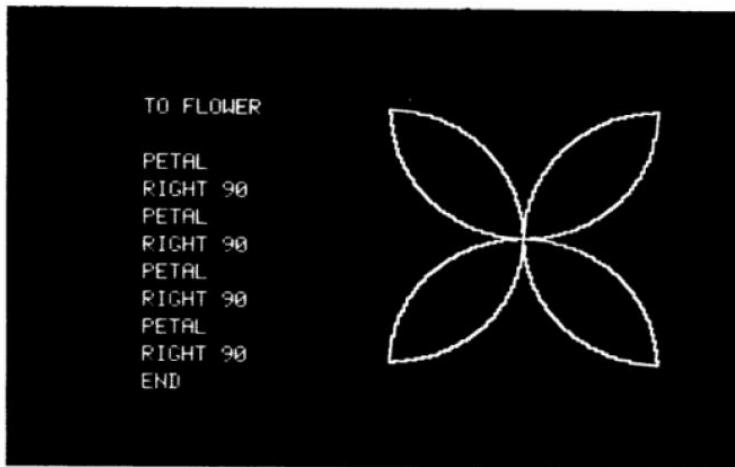
画一朵花的实例

可以用一段圆弧作为花瓣的一边，将它复制后旋转 180°，这 2 个圆弧便组合成一个完整的花瓣：

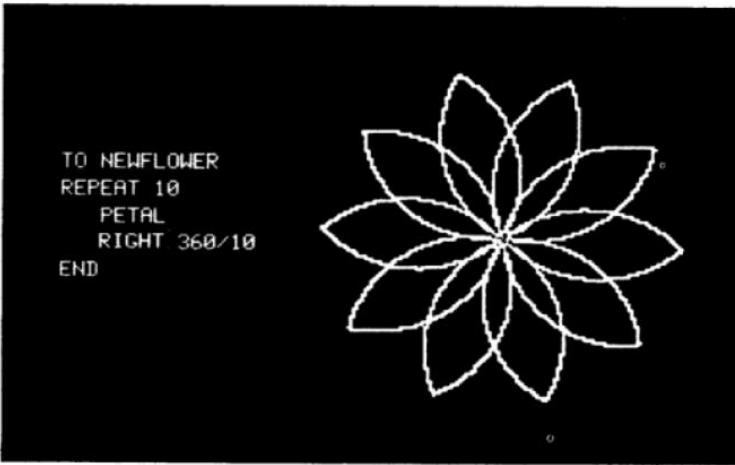
甚至圆形，在这个基础上，他们可以继续画一朵花，通过这种方式，学生就觉得  
自己可以掌握。



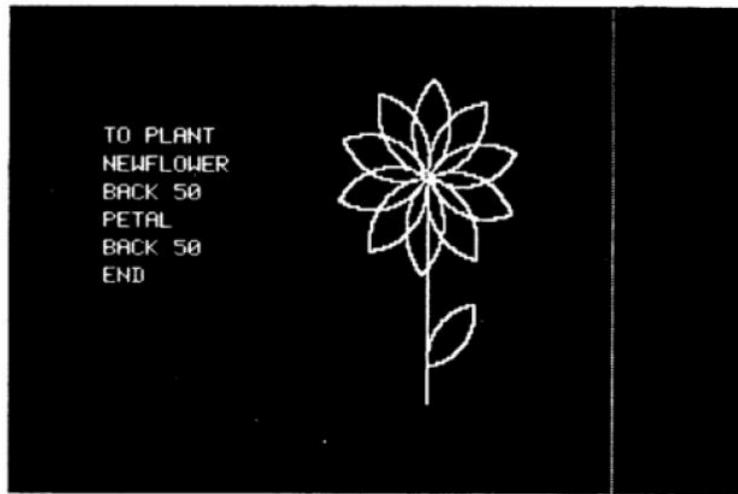
基于花瓣小程序，顺时针旋转 90°，重复 3 次，便可以画出含 4 个花瓣的花朵：



同样思路，基于花瓣小程序，花顺时针转 36°，重复 9 次，便可以画出含 10 个花瓣的花朵：



基于 10 个花瓣花朵的程序，在其中心点往下走 50 单位，画出一条直线作为花枝，在花枝旁加个花瓣作为叶片，便能画出下图：



乌龟几何游戏不仅让学生发挥无限创意，觉得自己可以掌握写程序画画，同时也学到软件功能复用的原理。

### 温伯格的教学实验

温伯格先生在 60 年代教程序员时，就发现很多人没有动手，导致上课学过的东西无法用在工作上。为了解决这个问题，他在教 IBMOS360 时，把本来的理论课改成一些工作坊教学形式。他还做了一个小实验，在一班讲课的时候强调要注意一些 OS 语言的常见问题，例如如何避免错误的空格导致编程失败，他发现平均下来还是有 83% 的错误。而另外一个课堂上没有强调这些细节的班级的错误是 87%。他还做了另外一个实验，对第二个班学生增加越来越难的练习题。到了第 10 题，10 个练习，17 个学生中有 13 个基本上就掌握了这技巧，没有再犯同样错误。从这两个实验可以看到，要有效学习，程序员必须积极动脑动手，不能单靠理论课。

### 参考 References

1. Gamma, Erich et al: Design Patterns: Elements of reusable Object-Oriented Software, Addison-Wesley 1994.
2. Parpert, Seymour: Mindstorms: children, computers, and powerful ideas, Basic Books 1980.
3. Weinberg, Gerald: The Psychology of Computer Programming: Silver anniversary edition, 2021(1971)

# Chapter 16

## 持续集成

依据 Thomas 先生的“每小步验证”原则，开发大型复杂系统，都应先拆分子系统/模块，先开发并测试子系统/模块、集成、再测试，按部就班地完成整个软件开发。

### 验收测试

验收要满足需求，需求要可验证，所以，通过相应需求的测试成为验收通过的必要条件。需求确定后，就可以规范验收通过的准则，开始准备验收测试了。

很多 QA 人员以为验收测试只须要做黑盒测试：

- 把开发出来的系统看成一个黑盒，开发人员已经做好本身的所有自测（包括单元测试集成测试等），然后系统要通过我们的系统测试用例，包括功能与非功能。

### 常见问题

当开发人员完成了所有模块的开发，开始进行系统测试的时候，往往已经到了项目（或迭代）交付预期的后期，这时如果发现大量缺陷，开发人员没有时间做好修复，即便修复后，也没有充足的时间做回归测试（重新跑本来通过的测试用例，确保不会因为开发人员的修改，而引起新的错误。）

### 改善方法

要避免这种恶性循环，减少发布延期的概率，QA/测试人员就要关注前面评审和测试（如单元/集成）缺陷排除率，以降低质量成本的概念，尽早在项目的前期暴露缺陷并及时解决，不要等到系统测试时才处理。所以不能单靠开发人员自觉做单元测试/集成测试。

从开发人员的视角，他不会想主动用测试找出问题。

最佳场景：开发完成，交给系统测试。没有发现缺陷，通过并结束，再开发新的模块。

“精明”的编码人员思路：“如果对测试/缺陷没有要求，多严峻的开发进度目标，我都能轻易达成”。

所以要做好验收测试，QA/测试人员不仅仅是做最后的黑盒系统测试，而是要从单元测试开始，设定好测试通过准则 (Definition of Done DoD)，然后交给开发人员执行，要看到所有测试，从单元测试开始，都完全通过，才算通过验收，以避免上面的问题。如果 QA/测试人员的技术能力有限，起码要明确所有测试的准则（测试用例）。

反之，QA/测试人员仅做最后黑盒测试，不能按时交付/交付质量问题，始终难以解决。

测试策略小建议：

1. 在项目一开始便要开始写测试用例（在设计、编码之前）。
2. 代码每次有任何变动时，利用持续集成系统自动跑测试。
3. 有了自动化就可以轻松实现回归测试。
4. 以上不仅仅是覆盖功能测试，也包括非功能，例如安全性、性能负载量等。

目的：尽早让开发人员获得反馈，以此驱动他们能随时达到一个可发布的状态。自动化让测试人员可以集中精力针对一些特有的测试，减少重复性测试的工作量。

## 测试自动化

与设计开发人员探讨他们的产品集成过程：

问：你们有没有用一些自动集成的工具，例如 Jenkins 集成工作自动化，原因是集成很花时间，现在也越来越流行持续集成、每天集成，所以如果靠手工集成、打包是很花工作量的。

答：没有，我们还是手工。

问：你们在手工集成之前怎么确保这些模块已经可以？

答：我们有测试。

问：做什么测试，可否举些例子？

答：我们开发完以后，会输入一些查询，然后看是否能正常使用。

问：我不是指整个系统功能的测试。例如，要通过你们 java 开发，有自动化单元测试，可以写脚本，如果你单元测试通过，就变绿，不然的话就红，你们有做吗？

答：没有，我们都是手工测试。其实我们现在到了产品升级、增加功能的维护期，记得当初首次开发的时候，我们开发人员是有用 Jenkins 自动化集成，但现在维护期我们就没有。

从以上对话可以了解到，很多开发维护团队基本就没有再做一步一步集成的过程，只是针对修改的功能，开发完就测试一下，然后便交到系统测试。

如果只是依靠对整个系统的集成测试：

- 不一定能找到所有问题。当我们改动了一些代码，却没有通过模块的单元测试，就很难确保模块的每一次改动都是正确的。所以，这将导致如果后续确有问题的话，就很难定位或者难以修复。
- 如果没有单元测试的话，你其实是不敢改动代码的，不知道改完以后对不对。

所以单元测试很重要，也因为单元测试高复用度，所以需要自动化才能节省大量手工测试工作量。

集成的概念，就是先写自动化单元测试用例，然后才写代码。通过单元测试才可以进入下一步集成测试，两个模块之间是否通。到最后进行总的系统测试，一步一步去做。如果团队像刚才那种情况，绕过了那些步骤，直接做总系统或集成测试。就会有很多隐患，到最终的验收测试才被发现，付出大量的缺陷修复代价。产品集成很重要，很花时间，所以绝大部分的团队如果注重软件质量的话，都会把产品集成自动化。改了代码之后，每天靠脚本来通过单元测试，比如晚上自动跑产品集成发现的问题，第二天暴露给开发人员修改，修改后再集成看看有没有问题出现。有些公司除了要求单元测试通过以外，也会要求通过代码的静态扫描，道理都一样。只有用这种一层一层进行产品集成的方式，才可以有效确保软件的质量。正如 Cunningham、Fowler 等敏捷大师所说，没有做好集成过程里的测试会产生很多债务，产品交付给客户以后，因为质量问题导致还债的代价就更高（详见附件某技术债务例子）。

## 持续集成

持续集成的道理很简单，每次提交代码做的修改，系统都可以自动构建整个系统，并跑通相关的自动化测，如果发现自动构建出错必须立马停下来修改缺陷。（详见附件持续集成步骤）团队要开始持续集成，起码要具备以下三个条件：

- 版本管理：所有的代码测试脚本、数据库脚本/构建脚本等都放在版本管理系统管理，不仅仅是代码。
- 利用工具实现自动构建（例如 Jenkins），虽然很多 IDE 集成开发环境都有这方面的能力，我们建议还是要用命令式（command）去跑构建脚本（像跑程序一样），每一次构建都能有详细记录（也随时可以重跑）。
- 持续集成不仅仅依赖工具自动化，还需要整个团队高度付出、参与和自律，每个人频繁以每个小步交付他的开发部分。James SHORE 先生在“Continuous integration on a dollar a day”文章里提出不一定依赖自动集成工具（例如 CruiseControl 是另一种类似 Jenkins 的集成工具）只要每个团队成员都做好自动化单元测试，每次提交代码都使用版本管理系统合并，也能做到持续集成。



不要以为每个开发人员每天提交（commit）代码到版本管理系统就算做到持续集成了，还需要：

- 每次提交代码要确保已经测试过，没有问题。

- 所有部署发布后的问题都能在 10 分钟之内解决。

不然就不算达到持续集成的基本条件。所以每天提交只是持续集成的第一步。

所以要做到持续集成得符合以下三个条件：

1. 是否频繁把变更更新到主干去。
2. 有没有对应的自动化测试，确保集成后的代码是能通过所有测验。
3. 如果构建失败不能通过测试，必须立马停下修正代码，直到测试通过。

有人会质疑为什么要这么频繁去更新主干的代码，自己分析测试好不也一样？很多持续集成的团队只做到第一点，部分能做到第二点，但能把三点都满足的很少。

如果只是自己测试，不能尽早暴露团队成员之间的代码冲突，如果可以利用自动化测试，频繁进行自动构建的话，就能及时暴露这些冲突。而且，因为每次变更的范围比较小，所以比较容易修复，反之，如果等写了几千几万行代码后再测试，就可能积累了很多的集成问题，难以有效修正。自动构建有版本管理系统支持，我们可以知道每一次变更的内容，也可以回滚到之前某稳定版本。

## 团队协作

当多位开发人员协作编码时，首先必须让他们做到“同步”，不然无法做到持续集成，先分享以下我的培训经历。

早上讲完结对编程的基本知识，下午让学员按结对编程做编码练习：

互动练习分四个阶段，每阶段都有明确的产物，计划总共花一个小时。（为了强调结对非固定，要更换，所以规定每个结对只有 15 分钟，然后换人继续 15 分钟编码）。

因为刚好有一半学员是一直做开发，另一半是做测试（公司已经推动自动化测试好几年了，所以测试人员也懂开发）。我们把开发归 A 组，测试归 B 组。

过了 20 分钟 A 组完成任务，B 组还没有完成。

为了两边同步，我们让 A 组等等，但过了 10 分钟 B 组还未完成，没有办法，我们让 A 组继续做下一个阶段，最终 A 组用了 1 小时 20 分钟完成所有阶段的编码，但 B 组到最后连第一个阶段都还未完成。（我们其实一直 B 组，最终到了 1 小时 45 分钟才终止。）

我们在旁观察，主要原因是 B 组有两位成员能力很差。

## 经验教训

分组原则：不仅仅看团队成员间能否顺利沟通，成员的技术能力也需要接近。

## 信息辐射器

当团队的能力相当，节奏可以同步后，便可以让团队自己设定奖励/惩罚机制，然后在大家都能看到的墙上，使用日历监控每天的情况。

敏捷大师 Martin 先生的建议：如果哪位成员提交的代码破坏了团队构建，他便要穿上一件脏且臭的 T 恤，上面写着“我打破了构建”(I Break the Build)，并且要在墙上日历当天贴上红点，如果当天全部构件都成功，贴上绿点。团队首个月还是红点居多，过了两个月后便逐渐反过来，变成绿点较多了。

## 持续交付

早在 90 年代，XP（极限编程）的创始人 Kent BECK 先生带领瑞士某保险公司的团队做软件开发，他们当时已经可以做到每晚发布了。越来越多的软件团队（尤其是互联网产品类公司）已经按照“尽量缩短交付时间 (cycle time)、尽快得到反馈”这样的思路在做了。

因为软件从完成编码到可以在客户投产环境成功部署，中间很多地方可能出问题，编码后必须经过构建 (build)、单元测试 (unit test)、集成/系统测试，最终在接近现场客户环境完成验收测试，才有信心能成功部署。

把软件系统分成多个子系统/模块，分到几位开发人员并行开发，各模块必须整合，并通过集成/系统测试，所以持续集成是持续交付的基础。

要做到持续交付，除了整个部署流程要自动化外，版本 / 配置管理也非常重要，不仅包括代码，也包括例如数据库 (DB schema)，脚本 (script)，环境配置 (configuration) 等，因为如果这些变动发生任何问题都可能会影响交付。

你可能会觉得持续交付太理想，难以达到。实际上有些面对全球客户的互联网公司（例如 Amazon）已经做到每天部署，但不要误会持续交付很容易做到，这些成功案例都已经经过多年的不断过程改进，并配合自动化工具，才能做到。

## 附件

### 持续集成

有很多开源的软件可以下载来用，选好你要用的持续集成 (CI) 软件后，你就开始要安装使用，希望利用工具来实现自动构建，跟安装其他软件一样，开始的时候会遇到困难，你可以在你项目的 wiki 记录下来，让其他人知道，避免以后重复错误。接下来，大家可以开始用服务器持续集成：

1. 当你准备好提交 (check in) 你的最新变更时，要确保它是否能正常运行。
2. 它可以正常运行并通过测试，你应该把你的代码从你的开发环境提交到你的版本管理 (Version control repository) 系统去，也看有没有更新。
3. 跑构建脚本和测试，确保所有过程都可以在你的电脑正常操作。
4. 如果你在本地构建成功通过，就可以把你的代码提交到版本管理系统。
5. 让持续集成工具自动构建你提交的更新。
6. 如果不通过，你要尽快在自己的机器上修改问题，返回第三步。
7. 如果构建成功，你就可以进入下一步：完成。

如果团队成员都按照以上的简单步骤，团队便有信心使 (开发的) 软件在任何电脑上，以同样的配置都能成功运行，一些持续集成的注意点：

1. 定时不断提交 (Check in regularly), 可以想象你写了很多代码才发现问题, 后面你会花更大精力来找出问题
2. 创建全自动化测试套件 (Create an Automated Test Suite)。以单元测试为例, 很多编码人员觉得写脚本式自动化单元测试浪费时间, 宁愿手工单元测试, 因他们以为只须要在写完代码后跑单元测试, 证明代码没有问题。当后面集成/系统测试发现缺陷, 只要修改代码的错误部分, 并能通过集成/系统测试便可。但修改后的代码还能通过本来的单元测试吗? 不一定。所以任何代码修改后, 必须再通过整个部署流程 (deployment pipeline), 里面包括单元测试。如果利用流程自动化实现持续集成 (包括单元测试), 便可节省用于手工测试的工作量。如果编码人员了解这个道理, 便不会再觉得自动化测试浪费时间了。(注 1)
3. 保持较短的构建和测试过程, 如果可以把测试和构建过程缩短的话, 就不会等到一大堆问题才要解决, 就像我们把复杂的系统分成几个子系统, 模块逐个开发的道理一样。
4. 管好自己的开发工作区, 不要只做好代码的版本管理, 配置管理应包括你的测试数据, 数据库脚本, 构建脚本, 安装脚本等, 因为每一块都会导致你的软件运作不成功。

(注: 为什么单元测试很重要已在《TDD 测试驱动开发》里说明。)

### 技术债务例子

员工: 总监, 我们过程改进是以高管的关注点出发的, 请问你有什么需求?  
总监: 我们的产品刚发布后, 客户做了软件安全性检测, 发现有漏洞, 这种情况就表示软件质量有问题, 客户感受也很不好。我们后面也很难去修正, 不知如何入手, 你有什么好建议?

我: 你们对产品有没有安全性的需求规范? 在开始设计或者开发时, 有什么对应措施? 我们都很清楚, 软件产品到了交付阶段, 已经把各模块集成在一起了, 如果这时候发现缺陷, 解决起来是很困难又费时的。你现在只知道有安全漏洞, 但不知道漏洞源自哪些模块, 就很难知道怎么去改, 改任何代码也可能会影响其他的系统功能。所以在交付后, 才发现缺陷的返工工作量是很高的, 可能要花好几天时间才可以正式的改正。但如果你们在开发之初, 就设计好对应安全需求的单元测试、集成测试、系统测试等的自动测试用例, 每次变更代码都在持续集成、自动构建里通过测试, 便可以避免后面这种验收才出现问题。

有些编码人员争辩说: “写自动化测试工作量很大, 我们也不是开发产品, 都是定制化开发。”但当他们理解到这些测试不是到最后跑一次, 而是每次代码有改动都要通过测试时, 他们才会理解不能靠手工测试, 必须自动化。

只有依赖自动构建、自动集成, 程序员才敢改动本来的系统的代码。如果没有单元测试、集成测试把关, 只靠最后对系统做功能上的测试, 无法知道中间的改动会不会导致一些本来良好的功能整出问题, 最后便导致客户投诉有安全问题。当系统已经是维护阶段, 因为大部分代码都不是维护工程师写, 如没有这种一步一步的自动测试来保证, 程序员根本就不敢改动任何一行代码, 软件开发行业称为“死”软件。

## 参考 References

1. Humble, Jez: *Continuous Delivery*
2. Shore, James: "Continuous integration on a dollar a day" ([www.jamesshore.com](http://www.jamesshore.com))
3. Martin, Robert C.: *Clean Agile - Back to Basics*



# Chapter 17

## 结对编程与同行评审

客户：我们的客户以银行为主，他们很注重质量，所以一直很注重评审。他们对需求评审、代码走查等也很赞同，也能找到缺陷，对提升质量有作用。但他们最困惑的是通过设计评审很难发现缺陷。

我：你听说过敏捷的结对编程吗？

客户：听过，也给客户做过培训，但是一般管理层都接受不了用两个人干一个人的活，所以几乎都没有团队在实际工作中使用。

我：当写完了几千行代码后，就很难在评审时要求改动。程序员会觉得又不是不对，测试也跑得通，为什么要改。所以代码的质量问题应在写的时候就避免，这是结对编程的原理。你可以把结对编程看成是一个提高团队之间互相交流的效率的工具。因为结对编程是在编码时针对具体问题集思广益进行解决，很容易落地，效果会比培训或评审好。

比如我跟另一位老师看一些团队的代码，他们都写了起码几千行。我们的结论就是：虽然代码的设计很有问题，但已经到了这个地步，除非重写，否则单靠修正部分代码并没有帮助。这也是设计评审常常难以做好的主要原因。而且大家都爱面子，如果在评审里，直接说他设计有问题，也可能会影响到大家的关系，所以在设计评审时难以找出缺陷，这很正常。

客户：你说的很有道理，但为什么这二十年没有流行？

我：有很多人不了解怎样做好结对编程。例如：

- 以为结对编程必须要找一位合适的搭档，人与人之间是否合得来最重要。
- 以为结对时只是在旁边看。
- 以为仅仅是培训，教同伴应怎么做。
- 以为只是看，作为人手编译器，只看代码是否写对，能否通过编译。

以上都并非结对编程的目的，也正因为这么多误解，才导致结对编程没有效果，后面就放弃了。代码规范很重要，你同意吗？

客户：同意。

我：代码规范本应是依据自己面对过的问题，形成清单，提醒自己要注意哪些问题——从实战中累积出来的检查清单，才有实用性。但很多团队的没有自己的代码规范，或只使用其他公司的。但如果团队结对编程，便能相互碰撞交流，

可以更好地区别出规范应包括的检查项。

客户：我也很相信结对编程有用，但有什么好的方法去说服老板，得到他的支持。

我：你觉得结对编程后主要的好处是什么？

客户：代码的质量应该有所完善。

我：同意，但能否具体一点呢？例如之前有什么质量问题影响项目客户。

客户：比如最近我们有个项目，测试都通过了，但是代码难以理解，读不懂。

我：同意，怎么解决这个问题？

客户：最终我们花费了大量时间对代码添加注释，还写了很多文档来解释。

我：后面加注释，添加支持类文档已经是亡羊补牢了，本该在写代码时就注意可读性。例如你觉得下面附件 A 与附件 B 代码，那种较好？

客户：附件 A

我：其实附件 A 和附件 B 都没错，但附件 A 比附件 B 容易理解。所以如果团队用结对编程，就可以解决写代码时的质量问题，确保可读性。减少后面再后补文档或者再重构的时间。如果你觉得代码可读性和后面难以维护是问题，让我们探索一下具体是什么问题，怎样解决。通常你们通过评审能发现多少问题（Bug）并在评审后修正、改善？

客户：几乎没有从评审找出可读性问题，我也带过一些初级的程序员，确实出现了不少这类问题，但项目时间很紧，我只能暴露那些最主要的大问题，这是可读性问题，确实很多时候就没有时间去处理了。

我：你可以估计一下，如果用了结对编程，要达到质量要求，后面可以节省返工。

客户：懂你的意思了，应该象前面 AP1 里那样，估算结对编程能为公司节省多少返工成本，并估计能带来多少回报。

我：是的，但估算节省时请注意不要误以为结对编程只能改善可读性，它只是其中一类。客户：如老板也赞同，应该怎么开始？

我：因大家都从未试过，可以先做一天培训，先感受一下应如何结对编程。因为没有培训的话，很多原理大家还是无法弄懂。

## 培训

### 结对编程目标

- 尽早识别问题，减少后面的返工。
- 促进知识分享和团队合作文化。
- 让两人可以相互协助。
- 同伴可放心评判对方的弱项。

先用代码实例让学员了解当前编码的不足，提出以下改进目标

- 促进代码规范。
- 提高编码可读性。
- 方便后期复用。
- 有单元测试（测试驱动开发）。
- 更好了解需求。

用小组互动练习，让学员体会以下原则

1. 一边编码一边说话。
2. 不断的同步问问题，比如这个方法是否应放在这个类上面。
3. 每个人都要带小本子和笔。
4. 最后应该不断的重用、复用。

工具

工作环境很重要，例如可以用一个大屏幕让两个人一起写和看代码：



或者用 Teamviewer 共享工具，让两位程序员各自用自己一台笔记本电脑：

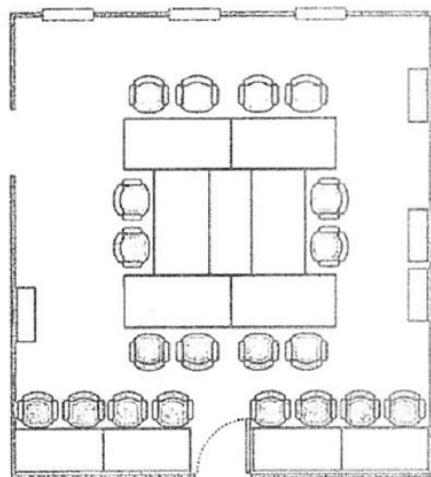


不一定全程都是结对，可以与单独交替



### 不断替换

如果结对编程总是由某一位编码，这容易开始，但是很难持续，这种互相分享知识的空间就越低。通常我们有好几种方式去对工作进行分工来做结对编程。例如最简单的是结对跟单独的交替或者共用跟私有结对等，也要考虑工位的安排。例如下图，有些座位可以让他们做到一起结对。另外有些是让他们私人做编程，减少干扰。



### 培训后的经验分享

培训的目的是让团队开始了解结对编程的精神：

- # 我们先定一些小目标，例如提高代码的可读性。
- # 改善变化，尽量让类可复用，而不是拷贝粘贴，然后尽量使用设计模式，而不是自己弄一个新的设计方法。

所以我们首先要有一个基本的代码规范，而且要求都写清楚。大家都了解以上原则后，可以让团队开始实战，通过练习感受结对编程的过程。很多时候学员都不习惯一边写代码，一边讲或者沟通，所以必须利用互动练习帮助学生提升这方面的能力。

### 总结

经过合适培训后，结对编程比评审更能有效地帮助团队写好代码，提高软件质量，并减少返工成本。

如老板赞同很多软件工程师经验少、没注意代码质量，导致后面大量返工，软件产品无法维护，我们除了可以以节省成本为由外，还可以用以下理由说服他：

- 不是所有代码都结对编程，而是针对重要代码结对编程。例如项目初期搭建架构和写偏底层一些的代码，以及同类功能用于当样板的代码。

- 结对编程结对编程，因两人一起解决实际难题，很适合用于以老带新的工作方式，效果比讲解/培训好。

## 附件

### 代码可读性例子

A	B
<pre>tmpDate="2013-02-01" for (k in 1: n) {   tmpDate=tmpDate+1   .... }</pre>	<pre>for (k in 1: n) {   tmpDate="2013-02-01"+k   .... }</pre>
<b>A</b> <pre>bonus = if (0==overwork) 3 else 6 monthPaid = basic + bonus</pre>	<b>B</b> <pre>monthPaid = basic + (if (0==colConDex) 3 else 6)</pre>
<b>A</b> <pre>time&gt;=(as.Date(today)-30) &amp;&amp; time&lt;=today</pre>	<b>B</b> <pre>(as.Date(today)-30) &lt;= time &amp;&amp; time&lt;=today</pre>



# Chapter 18

## 软件需求

客户：公司准备推行软件需求与研发分离。

我：为什么要分开？

客户：公司希望可以提高需求的独立性，也希望需求可以与研发相互制衡。以前需求是研发团队的一部分，很多时候就会倾向于从研发团队的角度来看需求。分开后可以更加独立，更多地从客户的视角看需求。

我们严格要求做需求评审，可否用评审结果来评判需求的质量？

我：我常常会问团队，需求评审找出的缺陷是越多越好，还是越少越好？如果有人说越少越好，我就会继续问是否 0 缺陷就是质量最好，所以我们难以单纯地用评审发现的缺陷数去衡量需求的质量。

让我先分享一下是如何来衡量过程质量。例如如何衡量系统测试的质量。看产品交付后有多少缺陷遗漏。如果没有任何遗漏的缺陷，那就表示系统测试做得很到位。如果往前倒退一步，可以从系统测试或验收测试发现的缺陷数来评判需求、设计或者编码的质量，如测试出的缺陷多就表示质量不好。但测试工程师无法判断那些系统测试的缺陷是因为代码还是需求引起的，所以如果要判断需求的质量，就必须在团队迭代回顾时，让所有成员（包括需求、设计、编码、项目经理等）一起分析缺陷源自哪里。所以过程的质量必须要从它的下游结果来判断。

我：你们把需求与开发分家以后有没有遇到什么问题？

客户：遇到问题可不少，例如研发团队会说，你需求写得太简单了，没写清楚，无法开发。而需求人员却说，你要写到多细才可以开发，你可以给我一个标准吗？甚至有些较极端的研发人员，明知需求有问题，但是还是按需求的描述做开发。

我问：你记得我们解读软件工程时，说需求与开发是不断优化、完善的循环过程。需求不可能一次性写好，而是先依据客户提出的一些初步的概念，内部与开发讨论，然后再与客户交流，慢慢细化的。但如果把需求与开发切开，就切断了这个交流的过程。

软件开发一直困惑于怎么写好需求，但一直都没有找到能写出完美的需求文档，开发出完美的软件产品的秘方。有些团队为了写好需求文档，花了很多精力，但后面开发出来的产品还是问题多多，甚至失败。所以在 2001 年时，17 位敏捷

大师针对当时瀑布式开发侧重文档的弊端，提倡敏捷开发，希望更有效地开发出客户满意的软件产品，而不是把时间浪费在前期文档上。敏捷开发模式也能更好地应对当时需求经常变化，导致后面项目延误和质量不好的问题。所以如果硬要将需求、研发分离，就是回到之前那种瀑布式开发的思路：花大量的精力希望写一个完美的需求文档。（敏捷开发的思路：软件开发的蓝图不是设计文档，也不是需求文档，而是代码本身。只有一个跑得动的代码，才是一个可交付、持久的东西。）

50 年代针对英国煤矿生产问题的研究（详见附件）证明，要提升生产率不能单靠科学化的过程管理来实现，还必须有团队内部的相互合作。采煤这类传统低科技行业尚且需要团队合作，软件开发就更依赖于知识工作者的团队合作了。

客户：我们公司领导希望利用标准化的流程跟一些相关的度量指标去控制整个过程。我就是负责制定这些指标的。但有个问题，比如我们认为研发做出来的产品应该得到客户的认同、让客户满意。所以客户满意度应该是个很重要的指标项。但研发组长就拒绝了这个指标，他说现在需求不是我们负责，我们只是做研发，为什么最终那个产品的满意度要由我们研发负责，因为我们控制不了需求的质量。如果我们要求需求人员或者产品经理对最终的客户满意度负责，体现在 KPI 指标，他们也会说，研发不是我们管，我们只做需求，开发做出来的产品客户不满意，为什么让我们买单？其实两边都有道理。如果我们希望通过一些指标，比如针对研发的指标，反而会对总体的效果有负面影响，比如客户满意度。但是大家也知道，如果客户对一个产品不满意，那开发肯定有问题。所以只度量某些过程没有意义，就好比如果评审或者测试的缺陷数越高，奖励越多，聪明的团队可以很轻松地变成百万富翁。人很聪明，针对指标，他们都会的方式达到要求。

客户：你说的很对，我们那些团队就很懂怎么朝那个指标作假。

我：所以要有合理的度量，尤其是和奖励挂钩的度量就必须是客观、项目间可比和全面。例如最终盈利多少比较客观，或客户满意度指标也可以很好地激励团队之间相互合作的积极性，对公司、对团队都有好处。但是如果只是用个别过程的指标去衡量，反而可能影响团队行为。

客户：公司觉得今年改革之前做新产品开发项目的立项太宽松，缺乏有效的管理和监督，这导致很多新的研发项目都是没有回报或者失败，浪费了很多资源。所以现在用新方式，所有立项的研发项目，都应该由中央委员会审批才能立项和开始做，我觉得这个倒是挺好的。

我问：你觉得好吗？

你听过苹果公司的故事吗？苹果公司本身就是由两人创办：一位是我们都认识的乔布斯，另外一位是沃兹 (Steve Wozniak)，他本来是惠普的工程师，在 75-76 年间研发出了苹果电脑 (Apple 1)，本来希望获得惠普高层的支持，就申请内部立项，但多次都被拒绝。乔布斯看到这个产品后，觉得很有市场潜力，就跟沃兹跑市场，最终获得 Byte Shop 老板 Paul Terrell 的首张订单，购买了两百套苹果电脑 (apple 1)。当时 Apple 1 都由沃兹手工焊接而成。创新产品都必须具备以下重要元素：

1. 了解市场需求的人员，乔布斯就是这个角色。
2. 研发工程师沃兹。

两者相互合作才把苹果产品做成。如果把需求跟研发切开，估计苹果的传奇故事永远都不会诞生。正是两人的团队合作，一起去满足个人电脑需求的市场才

会成功。从这个例子可以知道，要创新就不能用传统的工厂化流程，靠一些指标去管理。所以若单依赖一些质量指标去管理需求和研发过程，是不利于团队与公司的健康发展。

惠普公司很优秀，内部肯定不缺资深专家，评判的指标也应该很完善，为什么沃兹的内部申请立项，多次都被拒绝？惠普一直都是专注于做度量和测量仪器，但个人电脑是全新的概念，所以单靠现有专家的指标是无法鼓励项目创新。发明全新的东西必须是需求配合技术开发工程师，再有客户的接触跟确认，才可以成功实现。

客户：有道理，我们现在在新方式下几乎都没有什么创新了，都是按以前的方式做。

我：但这正是软件开发公司最致命的问题，没有创新，后面就会被竞争对手超越。

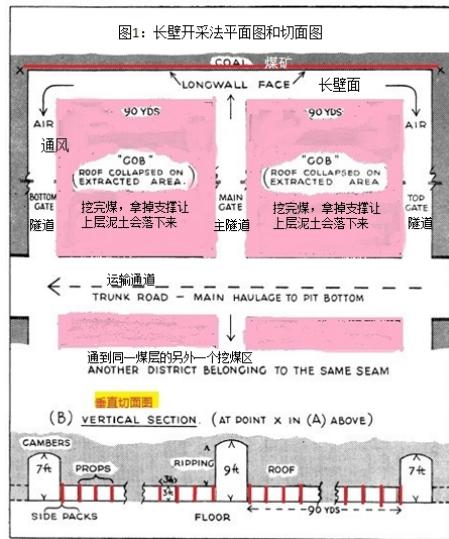
## 附件

### 英国煤矿生产问题的研究

在机械化之前，英国采煤是以小组作坊式工作，通常是两人一组，再加上一位做清理的助手。小组会直接跟矿场合作，专门负责某一块煤矿，小组以互补的形式做各种工作，自己管自己，独立性很高。因为采矿工作非常危险，所以选择伙伴很重要，都希望有稳定的伙伴关系，不会轻易换人。当某工人受伤或者去世，他的伙伴都会尽力帮助他的家人。这种小组作坊式工作方式后来被机械化生产线模式取代，但机械化生产却引起很多新问题。

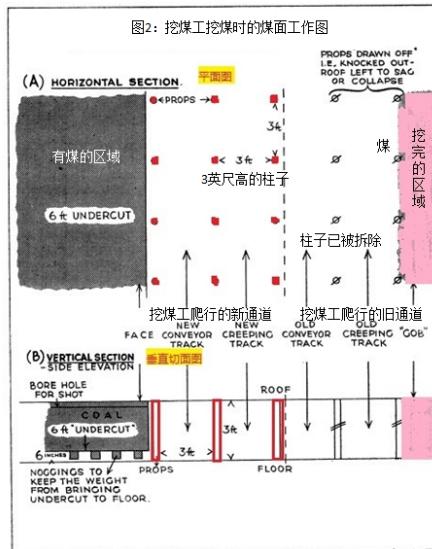
先了解一下长壁（longwall）开采法是怎么利用机械化大规模开采煤矿。

煤是古代的植物经过很多年的碳化累计下来的产物，英国的煤层一般不深，最多可能一米左右，上面覆盖着泥土。用长壁机械化采煤方式就可以大面积采煤。我们看看它是怎么操作的。



(FIGURE 1 说明：上图 A 部分是煤矿俯视图，B 部分是俯视图中右到左横虚线的切面图，都能看到是左、中、右 3 条隧道)

共一百八十米宽的长壁是为了大面积采煤，(看上图)为了通风和运输，会有三条隧道，之间距离九十米，中间隧道是比较高，大概有三米(九英尺)，左右隧道比较矮，大概两米(六英尺)高。这些隧道都是固定，让人或者那些机械也可以在中间运输。看上面(FIGURE 1)平面图，粉红色是已经采完煤的泥土。从下面的切面线可以看到中间的和两边两个隧道。每个采煤的煤层，只有一米高。也可以看见红色的柱子，柱子是用来顶住上面的泥土，防止泥土在煤挖空以后塌下来。



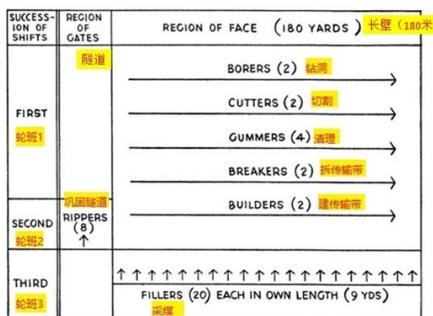
(图 2 说明: 与图 1 类似, A 部分是俯视图 (平面图), B 部分是横的切面图)

可以从上面那平面图理解整个长壁 (Longwall) 如何操作。右面就是已经采完煤的那些泥土, 我们用粉红色标注, 叫 GOB。右面有两条垂直的过道, 都是一米宽。前面两米宽的那条过道, 是之前已开采完的, 它左面会有一条传输带运作的通道, 右面是一条只有一米高的通道, 以便采煤工人爬过去工作。你可以想象环境多么恶劣。左面那个通道就是一条新建的通道, 同样它左面是传送带, 右面是矿工爬行通道。每一次采完右面的煤以后, 有工人会把那些柱子拆掉, 放到新的通道里面, 然后再依据这条通道挖左面的煤, 一直往左边伸。你可以想象在第一个图的左、右面九十米宽的长壁, 就是一个连续往上升的一条线, 矿工按分配的工种和轮班去挖煤。为了配合这种机械化挖煤方式, 工人就不能再用以前的作坊式小组工作。工业工程师设计了七个工种, 请参照下面的轮班表。每循环会包括三个轮班:

- 第一个轮班主要是做转动和切割和清理, 比如让那些切割的煤有空位落下来, 让后面那些采煤工人容易可以采到煤。通常第一个轮班都是下午班或者晚班。
- 第二个轮班就是巩固那个隧道和把那些传输带去做安装。也通常是下午班或者晚班, 这两个班总共就会大概有 20 个工人。细分到不同的工种, 比如有 2 位只钻洞, 2 位只切割, 有 8 位专门是管理通风隧道的巩固工作, 分工很细。
- 第三个轮班主要有 20 个采煤工人操作。第三个轮班只有早上班和下午班, 所以他们需要依赖前期技术人员做好准备工作, 然后自己按大概九米的范围去采煤。收入是按采煤量计算, 比如 20 个采煤人每人负责九米, 加起来刚好等于整个长壁的 180 米。

分工设计很科学, 如果都按正常操作, 每次循环可以采到两百吨煤。

图3: 长壁工人的岗位和在煤矿的工作



但是你估计, 这种机械化的大规模连续采煤操作会有什么问题? 生产力效果怎么样?

以上的工作关系很密切, 例如:

- 如果洞得钻不够深, 采煤工便采不到煤。
- 如果切割没做好会导致采煤工可采空间不足, 影响产量。
- 如传输带没有安装好, 不成直线就导致后面的传输停顿, 采煤工无法操作。

各种状况都可能发生，加上煤矿是天然的，本身就有很多变数：例如地层有断层，或者地下天然气等。各种人为因素或天然因素都影响采煤工能否正常采煤。加上采煤的工作环境很恶劣，导致旷工问题很严重。例如某采煤工因前面两个轮班工作没做好，导致煤太硬，采不了，就需要用机器、工具去采，很辛苦，效果也不好，导致他工作两天就放一天假，然后也不补班。有些轮班因为人手不够，剩下的采矿工人就需要在地下多做两三个小时，才可以把整个长壁做完。最后当有些轮班，采矿人数确实太少了，其他人也撂担子，导致无法正常开工。因为采用机械化大规模采煤的各种问题，有些矿场开始引入自主团队组织架构取代原本的轮班分工（下面称为传统分工）

表 1 是两个不同的组织架构，技术都一样，也是在同一个环境。左面用原本长壁的方式设计工作分工；右面的加上自主团队负责制分工。左面矿工只需要做一项简单工作，比如采煤，与其他工人没有任何关系（在前面已详细举例说明）。右面就需要组员合作，协调应对采矿的各种特殊情况。

从这个例子看到同样一个采煤技术，可以有不同的组织架构支撑，效果完全不一样。所以团队组织架构必须与技术部分相配合，不能单独设计工人工作。

	Conventional传统分工	Composite自主团队
Number of men 人数	41	41
Number of segregated tasks 任务数	14	1
Mean job variation for members 每成员的平均任务/变化数:		
Tasks work with 要处理的任务:	1.0	5.5
Main tasks worked 主要任务:	1.0	3.6
Diff. shifts worked 不同的轮班:	2.0	2.9

从下面表 2，可以看到右面的组织架构更灵活，生产率比传统的单一工作设计高。右面的自主团队架构更能适应变化万千的采矿、采煤环境。采煤的步骤安排，也必须按照按环境的不同来应对。但如果是左面的硬性单一分工安排，缺乏这种灵活性（除非是专业性强的技术工种，必须把工作细分，每个人做某一件事）。但在采煤机械化、批量生产这种技术没有这个限制，所以采煤工人经过一些学习可以兼任其他工作。

	Conventional传统分工	Composite自主团队
Productivity(%) 生产率	78	95
Ancillary work at face/(hrs per man-shift) 辅助工作(小时/每轮班)	1.32	0.03
平均后备人力/总人力 (%)	6	no
Shifts with cycle lag 轮班延迟 (%)	69	5
最长连续轮班数 (没有轮班有问题导致取消)	12	65
average per cent of coal won each day 平均每天获得的煤炭%		

团队组织架构除了让工人更好做好采矿工作以外，也可以更好满足工人的个人心理需要，包括互相支持，有团队合作的概念。但是如果按左面传统的工业化分工，就缺乏合作。导致很多采煤工只能单打独斗，孤立无援，导致心理压力很大。这个也可以从表 3 的缺勤率看得出来，会更容易无缘无故请假等，背后主因是分工设计导致工人心理压力太大，承受不了。团队互相协助的环境可以减轻这方面的压力。

	Conventional传统分工	Composite自主团队
Absenteeism (% of possible shifts)* 病工率(可能轮班数之百分比)		
Without reason 没有理由	4.3	0.4
Sickness or other 病或其他	8.9	4.6
Accidents 意外	6.8	3.2
Total 总数	20.0	8.2

# Chapter 19

## 客户参与

### 常见问题

某软件离岸外包开发中心，承接某政府医疗机构的 IT 系统维护工作，因为工作量变化很大，而且需要快速反应，一直都是使用敏捷 (Scrum) 开发方式，每两周一个迭代。但因为甲方提需求的代表都是医护专业，缺乏软件开发和业务分析经验，很多时候，每轮迭代提出的需求都很粗。例如护士办理病人住院：只发出一张表，列举了各种人群病人的入院要求，步骤与条件。

乙方本来只使用敏捷开发的用户故事 (User Story) 方式，“护士办理病人住院”只算一个用户故事，难以判断：

- 需求是否完整。
- 与其他功能的依赖关系。（例如如可能有高度传染病，便必须先做某些化验，如果阳性便必须去隔离病房）
- 各种异常情景应怎样处理。

为了避免开发出来的功能不能满足需求，必须靠乙方开发团队先做好详细分析，但他们反而不太懂业务，导致只能从技术角度分析，未能全面挖掘业务特性。

### 改进方案

- 使用功能点识别实体与行为
  - 便能容易识别出需求中，那些地方甲方没有考虑例如，除了新增，是否也需要有查询和删除功能
- 再利用用例 (Use cases) 与各种场景 (scenarios) 和对应系统页面原型图，与甲方代表讨论各种场景，系统应如何处理

### 改进效果

- 挑了两个医疗相关的团队做试点，在开发前预先用了这些方法与甲方充分讨论，三个月 6 轮迭代后，UAT 里的需求相关缺陷数平均降低 43%。

从以上案例看到，客户代表除了要参与，还要有具体措施做好需求，例如可利用功能点方法，用例与场景来挖掘与分析需求。

### 识别利益相关者

和杭州客户领导吃晚饭，领导就跟同桌的项目经理开玩笑说，“你们刚刚完成内部项目管理自动化工具，做调研的时候好像没有找过我？其实，我是其中一位经常要使用这系统的管理者，下面项目的监控、申请都是经过我，但我发现这个系统很不合用，对收集我需要的项目信息没有作用。比如没办法处理一些批准信息。”

从上面对话，可以了解如果没有全面识别项目的利益相关者，可能会影响到项目的成败。例如，我接触一些有些具备开发经验的需求人员，但他们通常只注意功能需求的技术细节。

问他们，“哪些是你的项目干系人？”很多人都会说，“甲方有协调员，要访谈哪些人都是由甲方协调员安排，我们听他安排。”所以为了避免未能识别所有干系人的风险，就要主动跟甲方一起策划。我们说沟通计划必须是甲乙方一起合作做出来，而且会牵涉甲乙方各个层次的人。

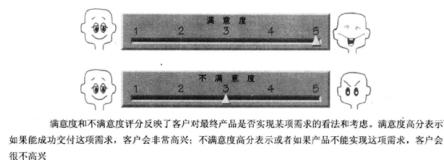
例如要听甲方出资人的需求，可能就要乙方的总经理出马，乙方的需求人员顶多可以跟甲方对口的项目组人员沟通。如何可以做好识别干系人，并制定沟通计划可详见附件。

### 用户故事卡

用户故事卡目的是让用户（业务）与开发沟通交流。

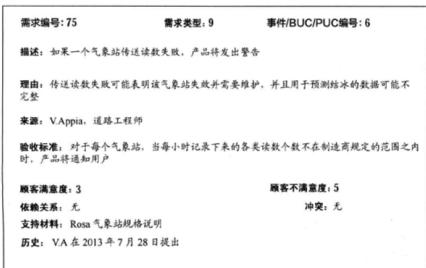
虽然 Kent BECK 先生强调用户故事卡片上的东西并非交流的全部，也不应该包括太多细节。但 XP/Scrum 的用户故事卡片未能全面包括需求的各元素，例如缺乏以下内容：

- 来源：每项需求都应可追溯到源头。
- 冲突：确保需求之间的一致性。
- 顾客满意度 / 顾客不满意度：用两个数比单纯用优先级能更全面反应客户声音。



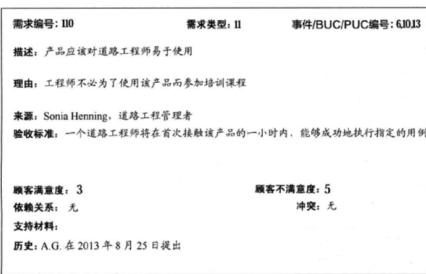
（如想多了解为什么要这样分，请看附件里的“客户声音：Kano Diagram”）

卡片例子可参考 ROBERTSON 夫妇的需求卡片模板：



白雪卡上的一项完整的功能需求

也可以用于非功能需求：



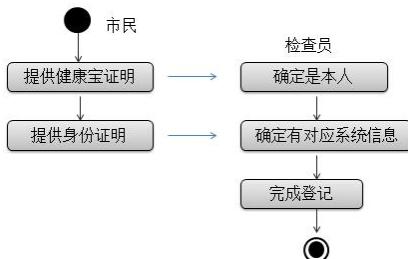
一项非功能需求，这是一项易用性需求

- 千万不要误以为卡上的所有信息都需要一次性与用户获取。
  - 挖掘需求永远是持续，并不断细化。
  - 卡片上增加了这些部分，可提醒我们不要忽略这些元素。

用例与场景

用户故事针对“做什么”与“为什么”（“What”，“Why”）；用例与各种场景针对“如何做”（“How”），所以它们之间是互补，没有冲突。

下面用大家都熟悉的简单系统“在核酸检测点做检查并提交结果”来举例说明，如果考虑不周全，也会导致需求相关问题。



正常情况：使用大陆身份证件时，用读卡器获取实名身份信息，与系统记录对应，记录信息也齐全，包括手机号码。

其他可选情况：没有大陆身份证，可用其他证件，例如港澳通行证、国外护照。

异常情况：市民身份信息不能用读卡器自动获取，需要手工输入，但检察员输入信息错误。例如护照号码错误，手机号码为空，被删掉或者没输入。

如果问需求分析师、产品经理有什么异常场景，一般都会回应“没有”；部分回应一些最基本的异常，例如“只有输入信息错误、报错，要求重新输入”，我便会以我的经验举例，说明必须要识别各种异常场景，才能算全面挖掘需求；识别正常场景是基本功，能识别各种异常场景才算做好。

我每次在北京做免费核酸检测，因为非大陆身份证，检测员都要用手机手工输入我的信息：姓名和港澳通行证号（因为系统已经有我的手机号，所以通常不需要再输入手机号），只要我确认以上信息都输入正确，通常都没有问题。后面不知道为什么原因？不再用手机，改用笔记本电脑配电子读卡器阅读电子身份证件和电子扫描器读二维码。

检测员不再要手工输入我的信息，靠我预先进系统填好所有信息，生成二维码，只要检测员验证了我的二维码信息与系统信息对应，便可完成。

一般正常操作是没有问题的。

但我最近在某采集点检查完，之后第二天未能在我健康宝正确展示出核酸结果。（开头以为出现了“混管阳性”，但之后几次核酸结果又恢复正常。）

我遇到的问题可能是因为那个采集点使用了笔记本电脑，信息先存在电脑，最后才批量上传到服务器。但因为非大陆身份证的情况，没有检查清楚信息是否都完备，例如，可能某些信息被检察员错误删掉了，但电脑没有报异常，最终批量上传时，服务器便无法识别我的检查记录。以上只是我的猜测，估计永远都不会知道真正答案，但无论如何，如果当初系统分析时能全面识别所有场景，应可避免这类问题。

如果使用银行交易系统概念，应可避免这类错误。比如我从自己账号转钱到另一个人的账号，中间会经过多家银行，我提交后，系统都会确保中间每一轮交易都已成功完成，并确认对方银行最终收到款，才算完成这个交易（通常前后经过几分钟）。中间任何环节出问题，都会撤回整个交易，能避免出现我那种不了了之的情况了。

如果信息有误或者不成功，导致最终无法提交上线，怎么处理？因总服务器本身或网络出问题无法连上，怎么处理？

如果要充分考虑各种异常情况，应仔细询问以下一系列问题：

通过检查正常情况的每一步并询问以下问题，可以发现异常情况。

- 如果这一步不能完成，或没有完成，或得到了错误，不可接受的结果，会发生什么？
- 在这一步会发生什么错误？
- 会发生什么事情，阻止工作到达这一步？
- 是否有一些外部实体可以打断或阻止这一步，甚至这个业务用例？
- 实现这一步的技术是否会失败或不可使用？
- 最终用户是否会不理解对他们的要求，或者错误地理解产品提供的信息？
- 最终用户是否会采取错误的行动（有意或无意），或者没有做出响应？

另外一类情况，有几十人排队等待做核酸，原因是服务器出了问题，停机了，排队人非常不满。工作人员也很尴尬说已经想尽办法用手机、电脑，但是都没办法。如果提前想好这种场景，就可以在本地把相关信息记录下来，等个服务器恢复后一次性上传，就可以避免刚才的情况。

除了正常，也可以考虑一些假设的场景，让干系人、客户可以更开放地考虑各种不同的情况，更创新。例如是否可以考虑像超市付款一样，自主处理？当很多人排队的时候，就不会因为检察员一个人忙不过来，导致排长队。

例如取登机牌也可以用机器，让乘客自主一组办理。其实做核酸也可以这样处理。

所以从以上案例看到用例 + 场景不同于用户故事，可以弥补后者。Kent BECK 先生特别提倡用户故事，因为当时很多客户都觉得用例太细、太偏技术，不愿意写，甚至也不愿意读；但客户大多都愿意用自然语言写用户故事，可以很容易触发客户与开发交流沟通。

但如果需求分析师只考虑现在业务流程的各种场景，开发人员开发系统，把所有的场景都覆盖好，是否便能成功？不一定，请看以下例子：

## 举例

你们都有申请过护照吗？比如我们护照更新，以前是要到柜台手工办理，如果我们把那个过程数字化，应该怎么做呢？

某国家的做法是本来的手工填写模板直接变成系统页面（每个输入与手工表格一一对应），申请人在系统里按本来模板填写并提交，上传个人新照片，也是经过系统，我有一次尝试用系统线上填电子表单申请，但因表单很繁琐，有很多护照原有信息都需要重新填写，光是填那表就花了我接近一个小时，最大问题还不是在我花时间填手工表，而是最后要上传照片，因为照片像素高的话就很大，需要很好的网络才可以传得上，如果照片像素小，便导致模糊不清，不能通过。最后，一个半小时后，我用尽所有方式都还是无法传上照片，我最终放弃了在线上提交申请，直接预约去在柜台做！

客户：有好方法解决这问题吗？

我：有，另外某国家的做法就很简单多了：

之前描述的整个过程只是把原有的手工步骤信息化，和原本的申请手续一样。但线上办理和在柜台现场办理不同，在现场你可以要求对方直接把照片给你，也可以要求对方提供老护照，但在线上办理，应很容易从系统里找到个人护照信

息，所以很多本来在柜台要手工填写的老护照信息就不再需要再填了。

客户：怎么可以简化整个过程？最困难应是照片的更新？

我：有些国家是这样做法，比如你申请续证，只需要填上老证件的基本信息，系统就立马能识别出本来的证件你确实有那个“旧”证后，便可立马提交申请。跟在网上购物一样，你确认过内容没问题，就在网上付费，然后打印申请表并在表上亲手签字，然后附上几张符合规格的照片，邮寄到政府机关。他做好新证件以后再邮寄回你。或者你自己到他规定的地点领取也可以。这样就能很简单地利用“低”科技解决方案，解决了刚才上传照片的困难。

从上面例子看到，我们应不仅是把那些本来手工的流程自动化，应该全局看要解决的问题本身，哪些过程自动化，哪些不应该自动化（比如传照片）。

甲方对怎么可以在线上做这个过程也没有概念，他只是知道，本来手工需要填表。乙方也不知道，他只是做开发，也不知道有什么方面可以不用 IT 方式，而用其它方式更合理。必须要一起探讨才可以有最好的解决方法。

所以作为业务分析师，你很可能要改变用户思考问题的方式，例如利用业务过程模型，配合场景与页面原型，与利益相关者一起探索问题的本质。软件系统必须为拥有它的人提供最理想的价值，构建软件系统本身（例如只是把现有流程自动化）不一定能解决客户业务问题。

## 总结

除了有客户参与团队，快速反馈，也要：

- 全面识别所有利益相关者。
- 利用功能点分析，了解范围，确保需求完整。
- 利用用例与各类场景，全面了解各业务事件都可度量、可测试。
- 利用需求卡片记录用户故事（因与干系人沟通，获取需求是持续逐步细化过程）。

## 附件

### 利益相关者

#### 利益相关者计划检查单 (Stakeholders Plan Checklist):

1/ 列出所有潜在的利益相关者 (List all potential stakeholders)

:1a. 类型 / 分组 (What are the base Segments)

:1b. 可否再细分 (Any sub-segments)

2/ 把她们分为 F (友好), I (忽略), U (不友好)

:Assign F (Friendly), I (Ignore), U (Unfriendly) to them

3/ 什么对他们重要

What is important to them?

4/ 学习目标 (Learning objectives):

: 针对某利益相关者，我们需要学习什么

:For each stakeholder, what will we need to learn?

5/ 怎样沟通 (How)

6/ 什么时间 (When)

7/ 抽样计划 (Sampling plan)

如何招募 (How to recruit)

如何能获得承诺 (How to get commitment)

[针对以上第一至第三项，参阅以下实例/解读]

### 实例/解读 (Examples / explanation)

1/ 全面考虑各类利益相关者 Stakeholders

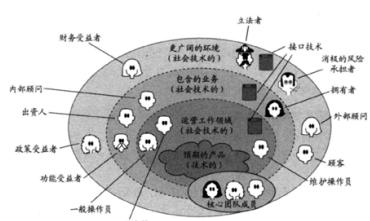


图 3-7 这幅利益相关者图展示了围绕最终产品的组织机构环，以及在这些区域中的利益相关者的类型。用这个图可以确定哪些类型的利益相关者与项目有关，以及自己需要代表哪些角色。

**出资人 (Sponsor):** 出资人为产品的开发付钱

**顾客 (Customer):** 顾客购买产品。必须对他们有足够的了解，理解他们认为什么有价值，所以会购买什么产品。

**用户 (User):** 确定用户的目的是为了理解他们所做的工作，以及他们认为哪些改进有价值。

在开发消费产品、大市场软件时，应该考虑用一个“假想用户”。假想用户是一个虚拟用户，他是大多数用户的原型。

**类型/分组的例子：**

未来笔记本电脑的潜在用户：

大类：

# 商业人士 (Business)

1. 媒体专业人士 (Media Pro)
2. 家庭用户 (Home)

**细分：**

# 主要用户 (Lead User)

1. 有极高要求者 (有挑战的 Demanding)
2. 潜在用户 (有潜力的 Potential)
3. 追求技术完美者 (技术流 Tech-Phobic)

客户矩阵 Customer Matrix 平衡数据收集计划 Data Gathering Plan				
	主要用户	有挑战的	有潜力的	技术流
业务	>销售VP >研发经理	>IT经理	>旅行社	>酒店职工 → 5
专业媒体	>动画制作者 >声音视频合成者	>录音室 >媒体实验室	>音乐家/歌手	→ 5
家庭	游戏爱好者	退休人员 (3)	业余爱好者 (2)	一家之主 (2) → 8 =18
基础群体		行和列的总和都等于 18		18个访谈应可以平衡地覆盖各分组。

## 2/ 策划包括哪些用户 (User inclusion strategy)

依据设计人员会怎么对应，把不同识别出来的利益相关者分成 **F、I、U**  
例：以针对笔记本电脑创新产品

**F(Friendly)** 友好，比如家庭用户、商业用户、媒体专员

**I(Ignore)** 忽略，比如忽略残疾人士

**U(Unfriendly)** 不友好，比如黑客、小孩（禁止他下载游戏、玩游戏）

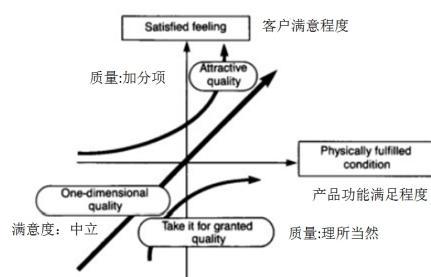
3/ 他们注重什么（主要关注点）

干系人	角色/概况	质量关注重点
商业用户	长期出差者 -坐长途飞机 -做演示 -保护某些秘密文档	-待机时长 -屏幕清晰度 -安全性
专业媒体	创造性工作；并要协同。 录音和录视频	数字带宽（声音和视频） 电脑速度和内存
家庭用户		

- 以上有那些在调研之前已知，其他有那些需要挖掘。

## 客户声音：Kano Diagram

可以用下图分析用户对需求优先级：



解读上下两个箭头应怎样看：

\* 下面的箭头代表理所当然 (Take it for granted)，如果缺乏，客户会很不满意，包括觉得是理所当然 (例如满意度：中立 1，非常不满 5)。

- 上面的箭头代表是加分项 (Attractive)，如果包括会非常满意，但如果缺乏会觉得不满意。(例如满意度：非常满意 5，不满意度：中立 1)。

所以“需求卡片”用两个系数：顾客满意度 + 顾客不满意度，能更好判断某功能属于哪类功能需求。

## 参考 References

1. BECK, Kent . D. WEST: "User Stories in Agile Software Development" , Ch.13 of *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*edited by F Alexander(2004)
2. COHN, Mike : "User Stories Applied "(2004)
3. MARTIN, Robert C.: *Clean Agile - Back to Basics*
4. ROBERTSON, S. : "Mastering the requirements process "(2/e) (2006)



# Chapter 20

## 团队协作

客户：极限编程 (eXtreme Programming XP) 强调团队要共同拥有代码，这条看起来很容易理解，但又很难理解。

当然我们一起写程序当然是一起负责，道理很简单，但为什么会有这一条原则？背后有什么原因？然后这章节会探索作为程序员和管理员，怎么才才算做好这一点？

我：讲需求与开发已强调团队合作很重要，但没有探索针对软件开发团队心理问题，之前只是用 50 年代的英国煤矿“长壁”大规模机械采煤的案例证明不能单靠正式的组织架构，加科学管理方式设计工作流程，来管理团队。因为实际的情况变化万千，也必须依赖团队成员之间相互合作，才可以有效提高生产力和矿工的心态健康。

我问：现在公司新人也很多，怎么识别谁是开发？

客户：很简单，不喜欢说话，不跟人家交流，每天沉迷在电脑前面那些人，肯定是开发人员。

我：一般人有这种概念，就是开发都是单独和富于创作性的，他们喜欢自己创作，不希望被打扰。员工专注做好工作，原本是好事，但如果做得过的时候，反而会产生负面影响。

客户：为什么？

我：因程序员越来越觉得程序是自己的作品，好像自己写了本书，在封面有自己的大名一样，但是软件和艺术作品不一样，如果有反对声音，觉得他作品不好，艺术家会认为是你不懂。但是软件程序不一样，它最后是在电脑运行，运行有错误就是有错误，所以导致程序员都会想尽办法证明自己的程序没错，问题不是出自自己。

一般人都会觉得自己比他人出众，如果做得好，觉得是自己棒，如果做错了，就是题目太难了，不是自己的问题。

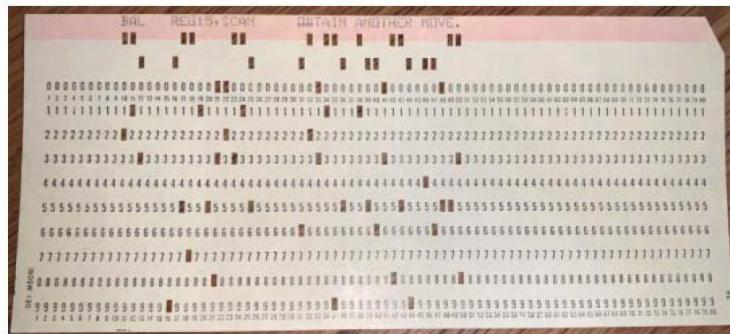
为了平衡这个心理，一般人会想尽办法找理由说明问题不在自己。

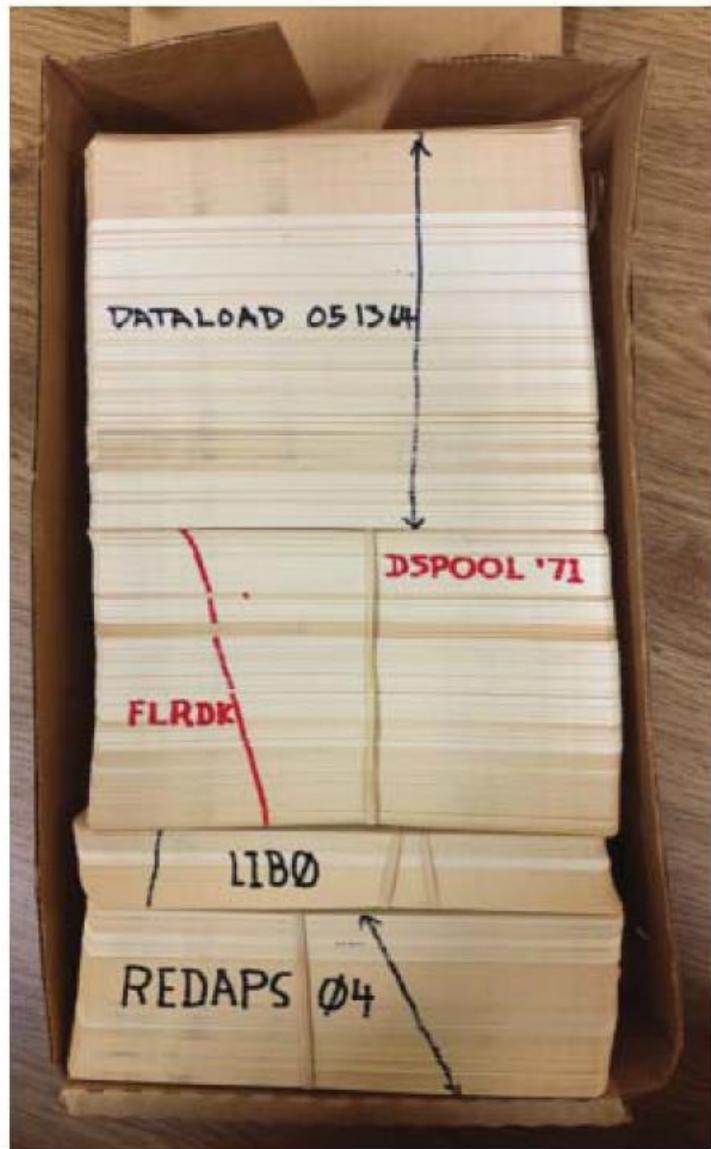
程序员的心理（比矿工）更加高深莫测，这问题从 60 年代软件开发的萌芽期已

经存在，不信可先看看以下故事：

各自负责自己开发的模块

六十年代，当程序还需要操作人员用机器把程序打在一堆卡片（一行一张卡片），然后利用读卡器把程序输入电脑的时代，如果程序有误，编程员会说是输入员出错，或者说是卡片的顺序错了。





你可以想象，如果程序员有这种心态，对整个团队的质量和交付都不好。所以我们每个人要抛弃拥有自己程序的心态，不要认为这是个人的作品，而是属于大家的，他会更能接受错误，对软件的质量有好处。

在实际工作环境中，大部分的软件都是有一组人合作写的，很少能靠单一个人完成。

## 团队合作

当项目需要多人合作，各有所长和所短，也出于更好满足需求，自然会形成一个团队合作模式，各人互补。当这个团队形成以后，团队成员会互相依赖，不轻易换人。例如某个团队预计公司会有计划解散这个团队，队员就开始寻找新的工作机会。最后这个团队选择一起离开，入职另外一家公司。新公司的经理觉得过来这些人都挺靠谱的，软件交付质量不错，当他有一些比较重要不能延误的开发工作，都会交给团队的其中一位。但他有所不知，虽然他是交给其中一位小李，但实际上工作是由团队内部相互合作完成的，不是一人工作。

从这个例子看到，一旦团队形成相互合作模式，它的作用要大于每个人相加，也正因为这个原因，团队会很慎重处理新成员的加入。假如有新人进来的时候会先告知新人，必须遵守团队的交付规则。例如后面加入了一位新人，他有3年开发经验的“老”程序员，觉得自己有经验，不需要遵循这一套规则，还是用自己的方式去开发，但越来越发现他的交付质量跟不上团队。最后有一次他交付了自认为是“良好”没有缺陷的程序给团队。为了避免再出事，他的程序被交给一位与他同时加入团队的实习生审查（这实习生加入后一直按团队规则交付，所以进步很快），他觉得很没有面子，最后他无法接受就辞职了。

这种情况不仅在软件开发团队存在，在合唱团或弦乐四重奏团也一样，非常依赖互相合作，所以在选择新成员方面非常谨慎，不要求个人英雄，更重要是要能跟其他人合作。

从以上例子看到，成功的团队是软件开发公司的宝贵资产，但管理者应该怎么去管理呢？要注意管理程序员这些知识工作者，不要用管理工人那种心态。比如下面这两个例子：

## 按时上下班

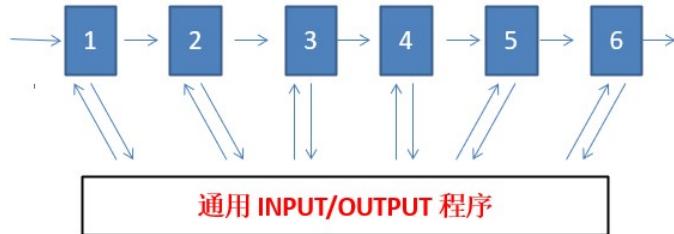
某大型项目中途换了部门经理，新加入的经理发现经常有程序员早上十点半才上班，他没有去问原因（其实有些程序员前一天晚上工作到凌晨两点钟），觉得这些情况直接影响团队管理，要立马改正，避免大家漠视公司规则而偷懒。他立马发一个通知要求大家要按公司规定时间上下班，并要打卡。

程序员立即集体反应：

完全按照经理的要求准时上班，也准时5点15分清理好桌面下班，并一起排队去打卡。但因为本来这些团队都默认按轮班制，有些早有些晚去对应电脑时间的限度（在60年代机器没现在发达，写好程序打完一堆程序卡片后，都是要排队让电脑按序运行。）经理发了这个通知后，生产率立马降低一半，很多程序员白天没事干，等机器有空才可以工作，然后电脑也按公司的时间运行，到下班时间就立刻停机。

六十年后的现代，也会看到同类故事在软件开发公司发生。

## 什么导致系统崩溃



有一个大型项目，希望把一个以前在老版本主机运行的应用软件系统更新到新硬件平台，总费时超过两年，团队平均人数在 10 位左右。这系统本来已经运作很成熟，所以整个开发最关键的部分就是更新硬件的接口部分，其他模块的开发相对简单。项目比较大，必须要分成六个阶段，其中有人负责这个重要的接口部分，其他人就负责原有模块的软件升级。过了一年半，一切的测试进展都很顺利，到了第五个阶段的验收测试就发现出了问题了，整个机器就停下来了。距离交付只有不到两个月的时间，大家就赶紧去找问题，仔细查看所有第五阶段的相关代码和接口部分的代码，测试之后和原有的比较没发现任何问题。

没办法，就在加入研究之前的第四部分，还是没有任何进展，时间也越来越紧了，大家很着急，又加了一些新的人来加入研究，因为开发时间比较长，有些之前阶段的人员都已经离开了。有一次，小张（他刚进入团队）与一位原本团队成员喝咖啡聊天，听到那位“老”员工回忆刚开始分工时，有一位特别厉害的程序员唐工，他非常不满意分工安排，他认为自己是团队里面最有经验最有能力的，他不满意接口部分的编程交给了李工负责。唐工整整有一周闷闷不乐，过了一周后，他就才正常平静下来。

小李听完以后立马有了灵感，就问唐工当时负责哪个阶段？哪个模块？那个老员工说，“他负责第二个阶段的那些模块。”这个小李回去立马查看第二个阶段模块的代码，很快便有发现，程序一开始就在内存里面读一个变量，然后把整个程序从那个接口调到唐工写的模块运行。原来唐工因为不满安排，写他负责部分的程序时，绕过李工负责的接口模块，让自己的第二模块直接对接，不用本来的接口。因他编程很厉害，做完这个动作后还让原本的变量原封不动，大家很难看出来。但他改变内存里那些记录，外围那些机器，如磁带机，因内存记录错误，就被搞乱了，所以一直到第四阶段都没有异常，但到第五阶段的时候就引起整个系统崩溃。

从上面的例子看到，软件协作并不是一件简单的事，外人很难看出其中做了什么手脚。如果我们分工时，没有真正让每个成员满意，也没有注意某些成员的心理不平衡，可能会导致灾难性后果。

## 技术团队的持续性

保持团队持续性也很重要，因为成员会离开，当处理不适当的时候，将会引起灾难。

例如公司小李很有魄力和经验，下面有 2 位实习生帮助他。但是为了这个程序，他一直就在离开家，到偏僻的现场做，他很想尽快回家，于是向经理申请：“我看到旁边有个小张，他也有经验，如果可以让他过来学习这个怎么做，我就可以被安排做其它项目。”

但碍于小张正在做另外个项目，经理不想调走小张，因为又要找到另外一个人代替。所以为了满足小李的要求，经理给他多增配一位实习生，但这对小李一点帮助没有，增加实习生只是增加他的负担。所以小李就再去对经理申请说：“不行了，你还是快点找小张过来帮我吧。”

经理最后的解决办法是增加了小李薪水 25%，希望他留下来，但过了不到一周，小李辞职了。经理最终没有其它选择，只能派小张接管这个项目。但因为他没有受过任何相关的培训，还是一团糟。过了几个月，这个项目出不了预期的效果，最后被取消了，对公司造成很大的损失。

从刚才的故事可以看到，一般经理以为薪水可以代替一切，其实不一样。对小李来说，加薪水就是希望他留下来，但是他就是不想留下来，所以这个加薪水反而变成他辞职的催化剂。从这个案例有 2 个经验教训：

- 每个项目每个岗位都应该有备份，如果有任何一个岗位不能有人代替的话，后面的风险极大，必须立马处理。
- 我们对程序员不能用一般管工人的心态，不是用钱就可以解决一切问题。

从以上的例子看到管理开发团队要注重成员的心理状态。这个道理其实对任何团队协作的工作都适用，但对于软件开发这种偏知识性行业则特别重要。因为每个开发都会觉得自己是高手，如果这块心理得不到平衡，他会用各种方式去平衡，甚至会对整个团队的表现产生负面影响。

客户：听完你这些开发团队的故事，我作为管理者是否应该放手尽量少干预，让团队自主管理，类似庄子那种无为而治道家思想就好了吗？

我：不一定，如果你只是放手团队自我管理，没有任何要求，团队很容易就会没有动力，后果也很严重，可以看看以下实例：

## 驱动团队改进

刚回国不久，还帮日本公司带领大陆的小团队做开发，这时有个朋友找我，说他的团队经常被客户投诉，请我帮忙看看。我就到这家公司，因为大老板以前是学校的老师，没有投入足够精力到公司的管理。本来这个团队的管理模式很自由，每个人和团队都非常独立，大家互相不干扰，很多时候帮客户做工作，客户觉得这个合作习惯了，也不想转变、换人，导致开发人员的效率和质量都不高。因为管理层也比较放松，开发人员也没什么压力，只要服务到客户不投诉就行。也没有任何规范和指标。我观察了一段时间，帮他们做一些辅导咨询，后面老板比较信任我的能力，于是邀请我来管理团队，我答应了。这时发现加入以后，管理他们更难了。比如我希望他们按我的要求有计划地加强管理，他们并不听，

甚至有时我觉得有些员工能力不够，需要调走，但他和客户关系好，客户立刻和我说，不能动这个人，否则我就不再继续签合同。这些开发人员已经老油条，知道用什么手段保护自己，尽量不发生任何改变。我没有办法影响他们的话，团队就无法进步，于是建议大老板说，我们废除以前根据主管打分决定员工薪水的方式，而是做成奖金池，用一些客观的方式来判断员工的贡献，而不是靠主管的主管判断。奖金是按开发出来的功能点数量，开发越多奖金越高，功能点规模是由独立的 QA 依据开发的软件客观算出。开始时，效果挺好的，因为有客观数据的衡量。但是几年后，他们熟悉了之后，就开始玩花招。听你说团队持续改进的方式，我觉得可以尝试，可以让团队有下一波良性的发展，提高他们的开发和创新能力。

### 总结

作为管理者不是插手、替代团队做他们的工作，例如编程、计划等，但需要设定高的提升目标，帮助他们自己提升，才能持续保持公司和团队的竞争力。不然这很容易被自然淘汰。在这个过程中，也要注意开发人员的心态，激励不能单靠奖金，尽量要让他们形成一个团队，相互合作的状态，对公司、个人都有好处。当这个团队相互合作的文化形成后，就可以帮助公司更好做项目，满足客户的要求。我看到很多团队成员在项目中间会被调走，这是常常影响到团队质量（与生产率）的主因。

**客户：**没办法，客户是上帝。比如他们有紧急维修要做，而只有一些有经验的开发能解决，就必须占用他们时间，否则客户不满意。

**我：**但这种临时抽调的做法，会影响到他们正在做那些项目，也会引起客户不满。

**客户：**你有什么好建议？

**我：**我认为可以考虑下面一些敏捷大师的建议，他们鼓励尽力保持团队不变，发挥团队相互合作关系的作用。

### 保持团队稳定

因为要让团队成员相互合作，变成一个有效的互补团队不是一件容易的事，而且是公司的宝贵资产。所以当项目结束后，不应该解散团队，而是要分派新的项目，让团队可以继续合作。但有时候，项目的规模大小需要不一定能与团队匹配，可以考虑以下建议：假如一个团队的生产力是 50 个单位，就可以安排以下三个项目来让这个团队负责，比如 A 项目需要 15 单位，B 项目需要 15 单位，C 项目需要 20 单位。尤其是如果某些项目后面突然间需要紧急加快速度，也可以很容易策划、改变分配的比例，这样会比平常增减人员简单很多。因为软件开发特别需要团队之间的沟通，如果你某项目需要加快速度，通常增加人是没用的，原因你会花更多精力让新加入的团队成员融入。本来团队要教他们，在这种后面增加人员的情况，收益还不如在过渡培训的投入。有时候项目延误，甚至不是团队能力的问题，而是因为有紧急需要，某些开发人员被调走，或者要坚固一些老项目或者其他项目的任务，导致这个开发人员同时要服务两到三个项目组和项目经理，导致开发人员无法专注做好编码工作。太多骚扰，总体效率、生产率也受非常大的影响。应该怎么安排呢？要避免这种临时的抽调。但确实有些客户的紧急问题，只有某些开发人员熟悉，其他人不懂。这种就回到

我们之前说过的，一个项目不应该有任何事情只有一个人懂，要避免这种临时抽调的话，就必须在本来开发的时候，团队不仅做好产品开发跟交付，也需要让他把知识传递给软件维护团队，到后面出现一些事故的时候，就可以让维护团队处理，不打扰本来的开发团队，开发专注开发，维护专注维护。就可以避免这种对公司不利抽调发生。

## 附件

### 平衡心态 Cognitive Dissonance

- 探索当行为和心理有冲突时，人们如何对应。
- 被安排做一些很沉闷的任务（例如：扭螺丝钉）。
- 试验者会收到 1 美金或 20 美金，要他告诉下一个人这个任务很有趣，很有意义。
- 试验完成后，再问实验者对任务真正的感觉。

1957 年实验，要求有些学生（被实验者）做一些很无聊的工作，比如反复钻螺丝、包装，后面给他 1 美金或者 20 美金。要他告诉下一位，刚才那个工作很有趣、很值得。实验后再问他们这个任务是否有趣？发现给 1 美金的人，大部分会觉得工作有趣。而给 20 美金的反而不同意觉得工作无聊。

原因：当你给他 1 美金的话，金额很小，他心里觉得不是被贿赂，所以会相信他自己说过的话。而当给了 20 美金，他觉得自己是收了那个钱才这样回答，否则肯定不会这么回答，所以心理上就会抗拒，不同意这个工作有趣。

分析实验结果：人会尽量说服自己“我做的是对的”，来取得心理平衡，所以即使他确实做的事很无聊，因为他跟人家说了这个事很有意义，他会尽量先说服自己，觉得这个是有意义，使自己心理平衡（收了 1 美元场景）。但反过来，当因为这件事收到 20 美金，他觉得这个说法不是自愿的，只是因为收了钱。反而就不需要动力去平衡自己的心态。

## 参考 References

1. Weinberg, Gerald : *The Psychology of Computer Programming*
2. Martin, Robert: *Clean Coder*

## **Part VII**

### **度量与分析**



当团队开始养成每天收集数据的习惯，在迭代回顾时分析数据，公司便可以开始准备度量与分析。（注）要做好度量与分析，跟过程改进一样，要针对目标，制定度量计划。本部分开头会以问卷调查为例，介绍度量计划的主要元素，然后会介绍统计分析，假设检验等数据分析的基本功。

并利用某银行软件维护数据分析案例，简单介绍数据收集后，做分析的主要步骤。

当数据会随时间变化，控制图可以帮我们识别波动是噪音还是信号，并识别当前的标杆范围（基线）。

注：请不要误以为度量与分析必须由管理层驱动。若要团队能从定性升到定量管理，并能持续，而且有效果，必须从团队开始，而不是靠管理者总体策划



## Chapter 21

### 做问卷调查



## Chapter 22

### 教小孩统计分析



## Chapter 23

### 假设检验



# Chapter 24

## 控制图

我：请问你们公司来年的目标是什么？

客户：希望提升 5% 市场占有率，总体的毛利也提升 10%。

我：公司只是定个百分比是难以监控。原因是大家都不知道本来是多少，所以后面难以判断是否达到。有些公司不仅仅用百分比来制定总体目标，也用百分比监控生产，你可随便看某月报。

例如某 7 月份报表里包括当前 7 月的数字与上个月的比较；与去年 7 月份比较；比较累计值等，但变化百分比有正有负，你看完这种月报能了解生产的变化情况吗？是变好还是变差？

客户：很多时候看不出来。

我：所以我们有以下原则：

要了解确实的数据变化，应该有数字。

但是仅有数据也无法“看”到变化，所以也需要把那些数按顺序画趋势图；比如你看下面 1987 ~ 1988 美国国家总贸易逆差每月的数据趋势图，就更好了解美国贸易逆差这两年的变化。

美国按月贸易逆差，1987 ~ 1988 (\$，亿美元)

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1987	10.7	13.0	11.4	11.5	12.5	14.1	14.8	14.1	12.6	16.0	11.7	10.6
1988	10.0	11.4	7.9	9.5	8.0	11.8	10.5	11.2	9.2	10.1	10.4	10.5

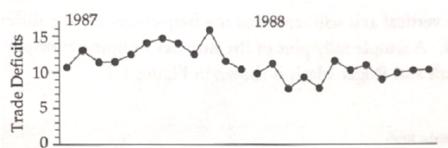


Figure 1.5: Monthly U.S. Trade Deficits 1987-1988

## 建立基线

有数据便可以建立公司基线（或标杆）作为以后参考。你看刚才两年的美国贸易逆差数据有变化吗？

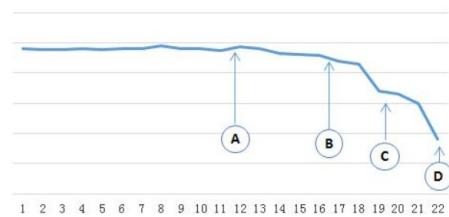
客户 1：有变化，比如中间 87 年 10 月份就特别高，后面 88 年的 5 月份也特别高，应该是不正常。

客户 2：我看不是，好像是后面那些数比前面低了。

我：很好，但要判断分析数据有没有变化，很难单靠主观识别，需要有工具帮助。

## 噪音还是信号

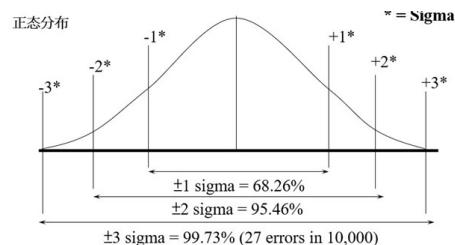
我：假设一家生产矿泉水的企业（通常一瓶水 500 毫升），为了确保每次机器装的水不会太多，也不能太少，每一轮生产都会随机抽五个样本，然后记录抽样的均值和范围（最大和最小）。下图是过去 22 天均值的变化：



肯定是后面比之前的下跌了，但是如果等到最后 D 点才行动就太晚了，已经跌了太多，但是在那个点开始有变？

这也是 20 年代美国电话公司要解决的问题：很多电话线路与设备都埋在地下，调配维修工作很困难，如何能尽量减少无用的调配工作？公司也利用生产线大量生产电话设备，需要调试生产线的各种设备参数，但工程师发现难以把生产线调试到稳定状态，如果按系数低了调高，或者高了调低，反而会越调越乱。针对这难题，Dr Shewhart（当时在 Bell Labs 工作）发明了控制图，用来区分是过程的噪音还是信号。你们高中的时候学统计学有听过正态分布吗？

客户：有印象，但全都还给老师了。

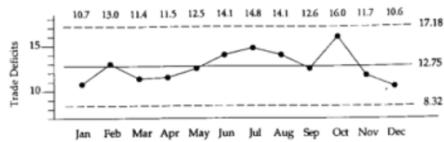


我：上图是正态分布，正负三个标准差包括了 99% 以上的面积。记得我们前面说过中心极限定理吗？任何分布如果随机抽样够多的话，它的均值就是接近正态分布。所以控制图按样本平均值的标准差，用方程式计算上限和下限，如果后面有些点超出了上下限范围（因上下限是按正负三个标准差出来，所以任何后面的点超出这范围，它发生的概率是少于 1%。）利用控制图帮我们区分那些

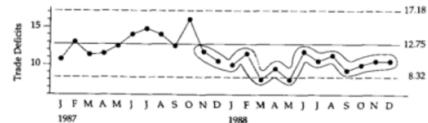
波动是噪音，那些波动（异常点）是信号。（想了解各种常用于生产管理的控制图，参考附件）

注意：虽然上面用正态分布来理解为什么选上下三个标准差为控制图上下限，但 Shewhart 先生的控制图方法没有规定或假定数据必须是正态分布。

回到两年美国贸易逆差数据，刚才不是说十月份超高吗，但是如果用控制图发现它还是在范围之内。如果我们多看两个月 -- 11 月和 12 月，发现贸易逆差又跌下来



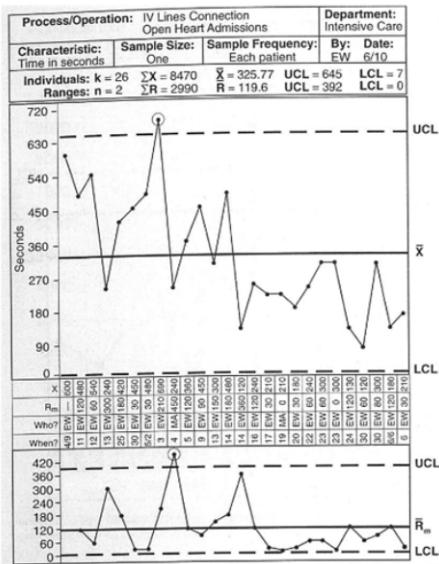
我们继续看，发现三月有异常点，后面在五月份也有异常点。如果我们再看后面月份的逆差数据，看见明显比前面降低。



客户：理解了。这个对于我们定基线有什么作用啊？

我：从刚才贸易逆差的数据，我们就不能把两年的数据的均值与范围（极差）作为基线，应该是以后面降低后的那些点，才能真正反应当前的水平和分布。

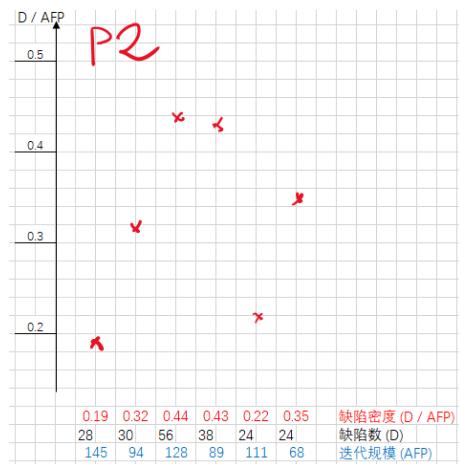
如果不是工业生产，不一定有这么多数据去随机抽样、求均值，怎么办呢？可以用 ImR 图（或 XmR），比如下图就是某个医院做外科心脏手术所花的时间的统计，想看看开心手术的时间是否稳定，因为外科手术特别危险，越长时间那个病人的存活机会就越低，也看到是有异常点。

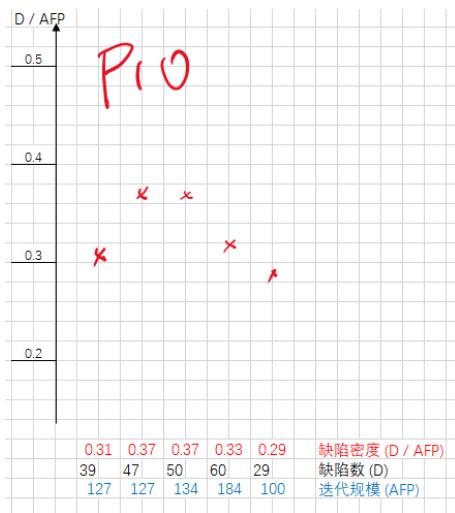


(ImR 控制图相关公式，详见附件。)

客户：可以用于我们软件开发吗？

我：是或者不是，比如一些已经进了维护期的产品都很稳定，你就可以用控制图来识别有没有变化。但是一些全新的过程可能变化很大，可能控制图就不一定适用。但如果你是比如客服收多少投诉，每天都是很稳定、连续的，应该可以用。下面是两个项目的迭代缺陷密度趋势图，请问你觉得那个项目的数据可以用统计图？





客户：右面 P10 应该可以，左面 P2 的数据太散了。

我：是的，例如，你会发现 P2 的控制图上下限非常宽，所以必须首先要知道是什么原因导致，完善后稳定，才可以用控制图。

## 控制图的应用

某企业非常关注交付有没有延误，每月都收集各个部门发生投产延误的次数，除以项目发布投产总数，统计并每月汇报高层。

从 2021 年初开始使用控制图，计算上限与下限 (0.98 , 0.93)，3 月份统计发现系数是 0.92 (0.92 是按时交付率，所以是越少越差)，超出了下线范围，改进小组启动根因分析。

每月的统计数是从 7 个模块（事业部）汇总来，改进小组利用 80/20 原则，识别出贡献最大的头两个模块已占总数之 73%，便专门调研这两个模块。使用问卷，面谈等，找出主因：

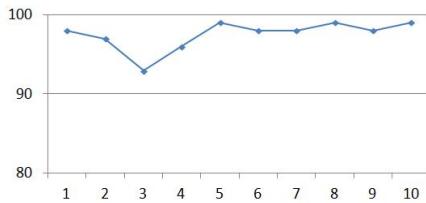
1. 新进项目经理较多，对里程碑管理、关闭条件等要求不清楚
2. 项目有点难，和用户沟通效果不佳

对应改进行动：

- 对项目经理的里程碑管理过程进行改进。包括：
  1. 培训（讲解项目管理工具的使用，和里程碑管理要求）
  2. 考核（明确如果出现里程碑红灯的考核标准）
  3. 预防（建立专项沟通小组，对存在进度风险的项目，提醒项目经理尽快推进项目经理或和用户提前沟通准备变更）

我问：效果如何？

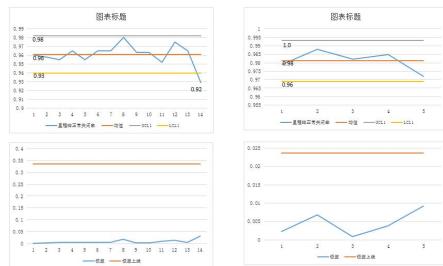
过程改进组长：请看下图今年到 10 月份的趋势图，从 3 月份的低点后面就明显改善了。



我：建议你用控制图方程式计算，从五月份到现在的统计图。

第二天，过程改进组长展示之前，之后的两个控制图：

下面左图是 3 月份 0.92 与之前一年多的控制图，右图是改进后 5 月份开始的控制图。



我：从左右控制图看得出改进后的下限从本来的 0.93 上升到 0.96。除了画图，也可以计算过程能力指数（Cpk）反映过程能多满足客户要求（Cpk 计算公式，详见附件）。

正与你说公司本来要求不能低于 0.93（客户要求规格），所以三月份之前的能力指数 Cpk = 1.0 但现在过程收窄了， $Cpk = (1-0.93) / (1- 0.96) = 1.75$  上升了。

如果公司因看到提现，也是收窄的规格范围，要求把下限调高到 0.96，你们的指数能力指数 Cpk 又变回了 1.0。所以 Cpk 可用来代表过程满足客户要求的能力系数。

过程改进组长：从五月后数据更新的控制图一直都很稳定，没有异常点，是否表示我们就没有在提升的空间，下限 0.96 已经是最佳状态？

我：不一定，你三月份分析时，不是说利用二八原则，识别出两个影响最大的板块吗？现在你的控制图是把所有板块的总数的控制图，建议你试试针对那两个影响最大的板块，画他们的控制图看看。有机会能看到一些异常点。

第二天，过程改进组长：确实看到有较大的波动，但还没有超出上下限，我们会继续观察。

以上实例帮我们了解：

### 1. 基线

从历史数据得出基线范围能帮助我们区分后面哪些波动是自然噪音，哪些是信号。信号表示过程可能有基本变化，需要更新基线。控制图帮助我们能做好这区分。

## 2. 过程能力 (Process Capability)

- 识别
  - 1. 过程的声音 (过程的自然波动范围)
  - 2. 客户的声音 (客户规格上限下限)
- 计算过程能力指数 (Cpk) 反映过程能多满足客户要求

## 3. 细分

- 不仅仅看总的综合控制图
- 从总图细分，细分后的控制图可能发现异常点信号 (详见附件有一个细分例子)。

## 总结

以上是如何利用统计图帮继续改地过程的典型案例，所以不要误以为控制图的目的只是为了控制不要发生异常点忽略了也要利用异常点看看过程有显著变化，是否启动根因分析，并采取纠正措施。

很多公司制定改进目标还是依赖主观判断，定一个“合理”提升百分比，使用统计图后便可以有当前过程的基线范围作参考，制定有具体范围的提升目标。

## 附件

### 如何画 ImR 控制图

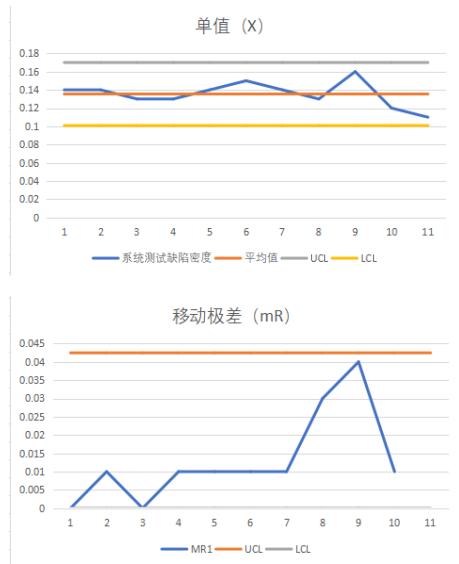
- 1/ 计算单值 (X) 平均数 ( $\bar{X}$ )
  - 2/ 计算移动极差 (mR moving Range) 均值
  - 3/ 利用以下方程式，计算 X 的上下限  
::UNPL= $\bar{X} + (2.66 * mR \text{ 均值})$   
::LNPL= $\bar{X} - (2.66 * mR \text{ 均值})$
  - 4/ 利用以下方程，计算移动极差的上限  
::URL=3.27\* mR 均值
  - 5/ 如果 X 或 mR 有超出范围，表示有过程变化的信号，过程不稳定
- Note:

$$\begin{aligned}\text{UNPL} &= \text{Upper Natural Process Limit} \\ \text{LNPL} &= \text{Lower Natural Process Limit} \\ \text{URL} &= \text{Upper Range Limit}\end{aligned}$$

例子：

系统测试缺陷密度	移动极差
0.14	0
0.14	0.01
0.13	0
0.13	0.01
0.14	0.01
0.15	0.01
0.14	0.01
0.13	0.03
0.16	0.04
0.12	0.01
0.11	0.01

- 1/ 计算单值 (X) 平均数 ( $\bar{X}$ ) = 0.135 [  $(0.14+0.14+\dots+0.11)/11$  ]
- 2/ 计算移动极差 (mR moving Range) 均值如第一移动极差为  $0.14-0.14= 0$ ; 第二移动极差为  $0.14-0.13= 0.01$ ; 10 点的均值 = 0.0128
- 3/ 计算 X 的上下限: 用上面公式  $UCL=0.135+2.66*0.0128=0.17$ ,  $LCL=0.135-2.66*0.0128=0.10$
- 4/ 计算移动极差的上限用上面公式  $UCL=3.27*0.0128=0.042$ ,  $LCL = 0$
- 5/ 下面 2 图已经加上上下限与平均线:



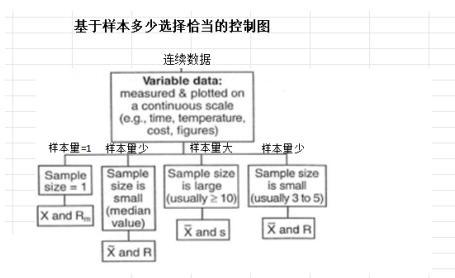
### 各种控制图

生产线用连续数据的控制图有两部分:

- 均值或中位数
- 标准差或者范围

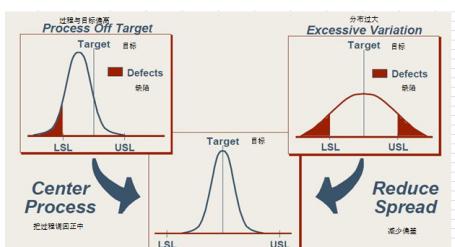
两个图都要看。通常一些数据本身的值有异常点的话，很可能它的标准差变化也会有异常点。

下面就是各种对连续数据的控制图方法，大部分都是按抽样样本的大小制定，比如抽样的样本数大于 10，我们就可以用平均值和标准差来做控制图。

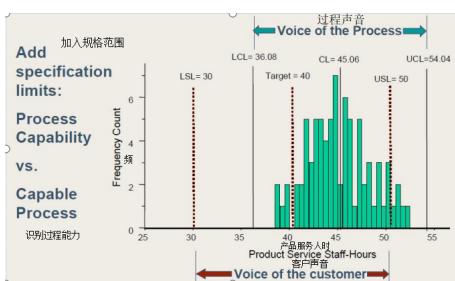


## 过程能力 (Process Capability)

过程能力反应能否满足客户要求：



中间的图就是客户的声音，USL 和 LSL 是客户要求的规格上下限，右上图就代表过程分布太宽，只有中间部分能满足客户规格要求，左图虽然过程变化范围满足，但偏离了，也不能完全满足客户要求。下图看到过程的范围是不能完全满足客户的规格范围要求：

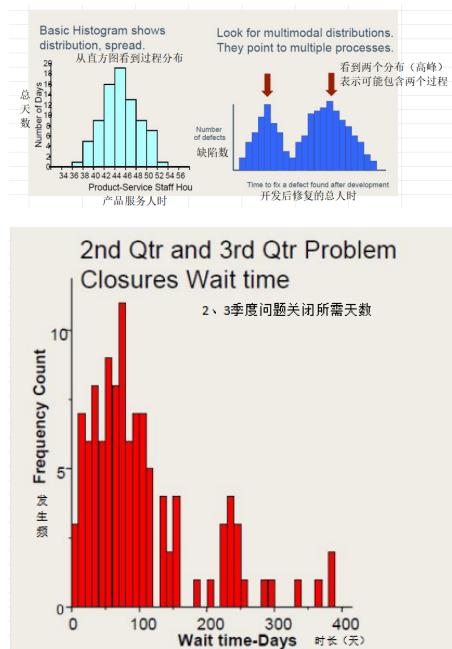


以下方程式可以计算过程能力指数。 $C_p = \frac{USL - LSL}{6\sigma}$ ，不考虑是否偏移，只考虑分布范围。

如果考虑偏移，计算  $C_{pk} = \min[\frac{USL - \mu}{3\sigma}, \frac{\mu - LSL}{3\sigma}]$  (选其中最少值) 指数越大表示过程越能满足客户规格 (客户声音)。

### 细分例子

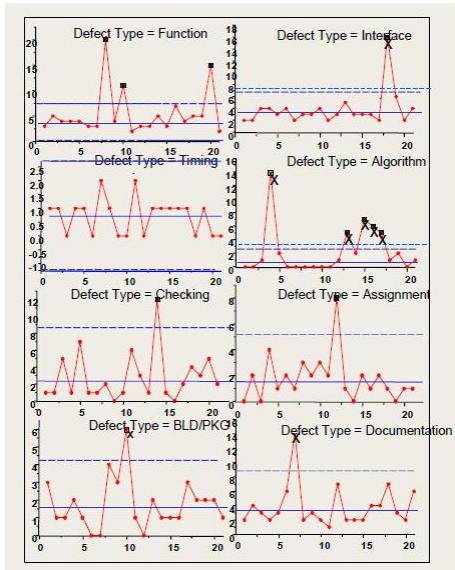
过程的分布柱状图也可以帮我们细分找出主要原因，比如下图右边这种柱状图表明这个分布可能有两个组成：



下表是关于客户的等待时间是多少天，可以看到 150 天以下和以上很可能是两个不同过程，所以我们分析的时候需要按这种细分，控制图也可以帮我们细分。

Component 组件	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	Totals 总数
Defect 缺陷数	12	16	18	32	22	16	23	35	15	27	16	25	20	26	20	23	23	36	22	27	17	471
Defect Type 类型	Number of Defects per Component 缺陷数: 类型																					
Function 功能	3	5	4	4	4	3	3	20	4	11	2	3	3	5	3	7	4	5	5	15	2	115
Interface 接口	2	2	4	4	3	4	2	3	3	4	2	3	5	3	3	3	2	16	2	4	80	
Timing 时序	1	1	0	1	1	0	2	1	0	0	2	0	1	1	1	1	0	1	0	0	15	
Algorithm 算法	0	0	1	14	2	0	0	0	0	0	1	5	2	7	6	5	1	2	0	1	47	
Checking 校验	1	1	5	1	7	1	1	2	0	1	6	3	1	12	1	0	2	4	3	5	2	59
Assignment 分派	0	2	0	4	1	2	1	3	2	3	2	8	1	0	2	1	2	1	0	1	1	37
Build/Pkg 构建	3	1	1	2	1	0	0	4	3	6	1	0	2	1	1	1	3	2	2	1	1	37
Document 文档	2	4	3	2	3	6	14	2	3	2	1	7	2	2	2	4	4	7	3	2	6	81

比如上表，某产品有 21 个组件，统计了每个组件的缺陷数，如果只是看总缺陷数的分布，平均在 22.4，看不出有什么显著变化，没有异常点，但是如果按缺陷的 8 个缺陷种类，画成 8 个控制图(下图)，就很明显看到有很多异常点：



## 参考 References

1. Wheeler: Understanding Variation The Key to Managing Chaos.
2. William A. Florac, Carleton: Measuring the Software Process SEI Series in Software Engineering 1999.



## **Part VIII**

**总结**



管理者应如何支持团队做好敏捷开发，为公司增值



# Chapter 25

## 创新：从个人到公司

### 引言

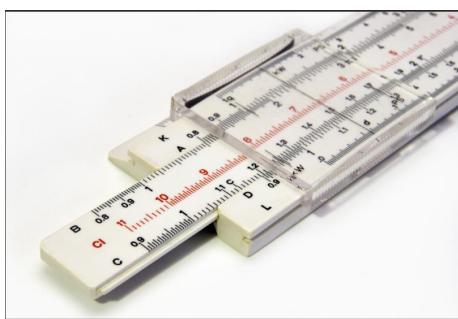
创新，顾名思义，是创造新的事物，提出有别于常规的解。

前讲过的丰 汽 公司精益产 (JIT) 的故事，精益产并由其创建，但其通过不断改善，实现了最少作，创造价值。即使团队迭代做根因分析，也都只属于解决问题 (Problem Solving)，并创新。

为什么有些公司和员，能够通过创新取得成功？下会先回顾个创新的元素，然后再探索团队如何创新，有什么关键成功要素。最后，从这些创新的故事中，探索与敏捷开发概念的关系。

### 400 年前的重大数学发明

如果你要你求 345 乘 456 等于多少，你现在可以马上用电子计算器甚至手机可以一秒钟得出准确答案。但在 70 年代前（还没有电子计算器），是怎么算这个乘数？如果手工计算，很费时。那时候读科生都使用计算尺 (Slide Ruler)，滑动计算尺中间那个可动部分，就可以得出答案。



对数是计算尺背后的原理。（如果不使用计算尺，也可以查对数表得出答案。）

对数是 16 世纪纳皮尔先生 (John Napier 1550 – 1617) 用了超过 20 年时间 ‘发明’ 的。

对数的简单例子：在草稿纸上就能算出 “ $10,000 \times 100$ ” 的结果，同时，我们也可以通过将 “ $10,000$ ” 和 “ $100$ ” 的 “ $0$ ” 相加，得出答案为 “ $1,000,000$ ”。也就是说，把 “ $10,000$ ” 看作是 “ $10$ ” 的四次方，把 “ $100$ ” 看作是 “ $10$ ” 的平方，将四次方和平方的 4 加 2，即可得出答案。纳皮尔注意到了这一数字法则，总结出了对数的概念。

计算 “ $10,000 \times 100$ ” 的话，使用乘法会更快，但如果数位数较大，需要手动计算时，使用加法运算会更加简单。纳皮尔先生按照这思路制作出对数表，简化了计算。

看到这里，也许有读者会想 “这不就是指数运算的法则吗？” 即是说，按照指数运算的法则  $a^m \times a^n = a^{m+n}$  来思考的话，

$10,000 \times 100 = 10^4 \times 10^2 = 10^{4+2} = 10^6 = 1,000,000$ ，如此，则可导出答案。然而，在纳皮尔时代并没有指数这种书写方式，指数的概念也不明确。纳皮尔的伟大之处也正在于此，他在没有指数这一概念的情况下发现了对数，并将其归纳为一个体系。

可以想象 16 世纪欧洲，数学研究还是早期，能发现对数把本来复杂耗时的乘或者除变换为  $\log$  后变成加和减计算本身已经是不容易，为了体现这个做法他还单靠个人努力，用了 20 年时间手工计算出相关对数表。现在我们可以还是可以找到他当时算出的对数表，用现在的计算机验证一下他的八位数头七位还是准确的。

纳皮尔先生的对数表：例如：第一行列出  $18^\circ 30'$  的正弦值为 0.3173047，对应数值为 0.111478920  
550px| 无

当时为什么要计算这些复杂的正弦相成？纳皮尔先生也研究球面三角学，使用他的纳皮尔公式角度与球面弧（距离）之间关系。16 世纪欧洲国家主要靠航海发现新大陆来发展。但海员在大洋中如何知道自己的位置，船员在大海中只能依赖天上的星星，用六分仪估算船经纬度。使用纳皮尔公式，便能估算 A 和 B 点之间地球球面弧长。

$$\cos \theta = \cos \varphi_A \cos \varphi_B \cos(\lambda_A - \lambda_B) + \sin \varphi_A \sin \varphi_B$$

例子：北京和巴黎距离：

A: 北京中心位于北纬  $39^\circ 54' 20''$ ，东经  $116^\circ 25' 29''$ 。转换为纬度 39.905556 经度

116.424722

B: 巴黎转换后纬度 48.8583 经度 2.29451

$$\begin{aligned}
 \cos \theta &= \cos \varphi A \cos \varphi B \cos(\lambda A - \lambda B) + \sin \varphi A \sin \varphi B \\
 &= \cos 39.905556^\circ \times \cos 48.8583^\circ \times \cos(116.424722^\circ - 2.29451^\circ) \\
 &\quad + \sin 39.905556^\circ \times \sin 48.8583^\circ \\
 &= 0.767103 \times 0.657923 \times \\
 &(-0.40881) + 0.64152 \times 0.758084 \\
 &= 0.28
 \end{aligned}$$

$$\theta = 1.287$$

(rad) AB 距离 = 地球的半径  $6,378KM \times 1.287 = 8,208KM$  (实际距离是 8,217KM)

当时没有电子计算机，他发明了对数解决这类乘除难题（对数不仅仅用于航海，也帮助天文学计算）。对数帮我们完成复杂计算超过 350 年，到电子计算器出现才退出舞台。

## 创新 Creativity

不是每个人都能像纳皮尔先生做出创新发明，为人类做出巨大贡献，但他也是普通人，为什么驱动他花 20 年精力独创对数？

Robert FRITZ 回忆：“60 年代，当我在 BOSTON Conservatory 学音乐作曲时，开始思考学的不应仅仅是位对、和音，更要了解音乐大师们的创新过程。”

他毕业后继续做音乐创作，一次参加创新专业人士聚会，包括作家、画家、音乐家、建筑师等，他发现这些人在针对本身专业的创作能力都很厉害，但都没有想到用他们的创新能力提升自己的生活。

他便开始举办培训课帮不同背景的人提升创新，让各个行业也可以学什么是创作/创新。几年后，他把创作的重点写成了一本书（见 Reference 参考）。

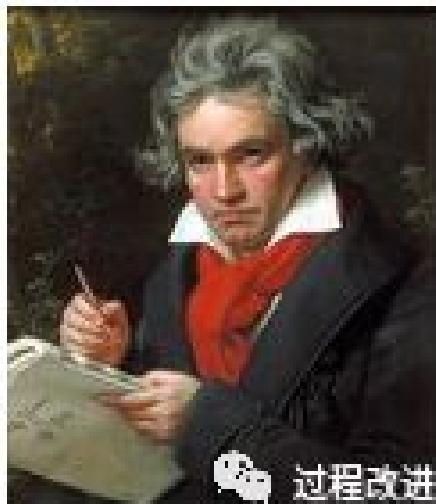
Robert FRITZ 在他的书中，以贝多芬为例，说明创作并非 Problem Solving（解决问题），而是要有一个很高的目标/理想，然后不断地尝试比较，最终达到创作目的。

## 创造的精神 Spirit of Creating

创作不是由他人驱使，而是出于自身创作的欲望，要创作世界上最优秀的作品。一个作家觉得自己的创作有价值，会用尽精力努力创作，最终希望自己的作品以后有自己的生命力，受他人赞赏。贝多芬追求完美，拥有极大的魄力，不断完善，才取得超人成就。

贝多芬对音乐创作的远景目标——希望创造前所未有的作品（他觉得当代的作品，甚至包括他老师海顿和音乐天才莫扎特的作品，离这远景还差很远）。由这崇高的目标驱动，加上他自身的音乐演奏和作曲能力，不断突破，甚至耳聋也阻挡不了他的创作力（从 25 岁开始听力减退，到 45 岁已经完全失聪），让音

乐创作升一大台阶，深远影响以后的作曲家。



贝多芬 (1770 – 1827) 的音乐创新可以说是前无古人后无来者，深深影响着后面两个世纪的作曲家。

我们都熟识他的命运交响曲，其实他从 1795 年到 1827 年去世前出版的每一首作品，无论是交响曲、室内乐或歌剧，都是经典。他是完美主义者，例如《田园交响曲》的创作时间不少于 3 年，从他手稿得知，首乐章某主题他修改了不下 12 次，所以与巴赫，莫扎特，舒伯特比较，他的“生产率”不算高，平均每年出版 5~6 个作品。

他出版《命运交响曲》、《田园交响曲》时 (1808)，已经出版音乐作品十三年，包括 23 首钢琴奏鸣曲，9 首弦乐四重奏。开始时，作品类似海顿、莫扎特的风格，但是到了中期，如《命运交响曲》、《田园交响曲》，已经远远超出他的前辈和同辈，启蒙音乐家从古典期发展成浪漫期新风格。

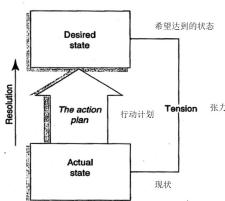
例如，低音大提琴一直在乐队里只担当伴奏角色，在命运交响曲第三乐章，贝多芬让低音大提琴先弹出轻快的主题后，大提琴接棒演奏同一主题，之后让中提琴接棒，最后又小提琴接棒。这种交响曲风格是前所未有的。(他写作时，还特意咨询当代杰出低音大提琴师 Dragonetti，问他是否技术上是否太难，是否超出乐手的能力。)

例如，第九交响曲，在最后第四乐章加入四声与合唱（传统交响曲只有乐队演奏）。

晚期的作品（如钢琴曲、弦乐四重奏等），又呈现出与中前期不同的另一种风格。例如他的 Op133 弦乐四重奏 Große Fuge 由于思路远超于当时的水平，不被当时的音乐家接受，觉得作品有问题，有些甚至说他太老，不听聋了，并疯了。但贝多芬依然故我，说“这曲是为未来写的！”

他的音乐创作一直没有停下来，例如他的最后弦乐四重奏（第 16 首）Op135 在 1826 年出版。贝多芬一生写了 32 首钢琴奏鸣曲，与莫扎特和他老师海顿不同，贝多芬每一首钢琴奏鸣曲虽然都是奏鸣曲式，但都有独特创新，突破奏鸣曲式，变化万千。

FRITZ 发现这些大师有共同特点：把握自己命运，追求个人目标的主动心态。很多人也很有才华，为什么他们大多数都不能成为大师？他用下图解释。除了要对未来有远景目标外，了解现状同样重要。如果不觉得有差距（张力），就没有改进的驱动动力；有些人有理想，但缺乏计划与行动，便只是梦想。



这创新动力模型不仅仅适用于音乐创作。例如，纳皮尔先生为了解决正弦相乘的困难，发现可以换成对数，使乘简化为加 (VISION)，为了可应用，还花了毕生精力，计算对数表，也是从张力，化为行动 (ACTION) 并创新的例子。

## 公司如何创新

这公司创作的产品在你身边的人中有大于 50% 机会都有使用，包括各种年纪，公司产品功能广泛，包括听音乐，打电话，电脑/平板，网上看视频，和日常工作。请你猜猜公司名？

苹果公司 (Apple)

我和全世界千千万万人一样每天都在使用苹果产品：iPhone，iPAD 平板电脑和 iTune 网上搜索音乐歌曲，要了解苹果的成就，便不能不谈乔布斯 (Steve JOBS)。他一生充满传奇，很多故事，可读“Steve JOBS”一书多了解。让我们先回顾苹果如何成为大众音乐市场巨人的故事。

====

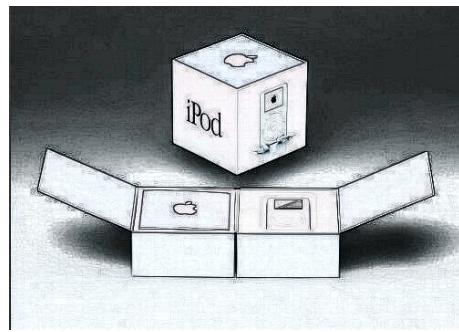
## 音乐播放器

80 年代末开始出现 MP3 格式，可以把歌曲数据化，市场开始有一些可以随身携带听音乐的小设备，但因为技术有限，都不理想，乔布斯本身也很喜欢音乐，就看准这市场，希望把苹果电脑定位为个人各种电子设备（包括音乐播放器）的集合点 (Hub)。首先要解决播放器方便携带，容量够多。当时还没有合适的小型显示器，锂电池和小硬盘。过了几个月开始找到合适，例如东芝刚研发出一款超小的 5G 硬盘，只一英寸直径，像个大银币。他们比较当时市场常用的播放器，发现都不好用。容量也不足，也难以按自己喜好排列歌曲顺序，如果要从 CD 把歌曲下载到电脑，并要按自己喜欢的顺序来让播放器去播放，其实是挺复杂的过程。但是如果把那些功能都放在播放器操作，但播放器屏幕小，太困难了。所以他一直非常关注产品易用性。当乔布斯第一次见到内部软件研发工程师开发的刻录软件，听完他们展示后，他就站到白板上画了一个框，说“你们设计那个软件，就是要把歌曲拉到那个框里面，点击一个按钮刻录就行了。”当时软件工程师看完以后吓呆了，觉得要做很不容易，但也恰恰是从工程师的

不会从用户的视角看问题。所以后面对整个播放器的软件，都需要给一个不懂的人使用，让她可以在三个步骤内，不用看说明书，完成任务，这是他的测试要求，做不了他不会接受开发出来的软件。播放器设计也要创新，比如用家可以滑动面板的轮来选歌，他只要在这轮不断的滑动就可以跳到要选的歌曲，很方便。iPod 也体现了苹果设计的概念 - 简洁。产品的设计选择白色，包括耳机也要白色（当时，大部分耳机都是黑色），还有连充电器整也是白色。设计师 Jony IVE 解释：“白色可以让他觉得这个产品很高贵，跟那些其他廉价的播放器有极大的区分，会觉得设计有艺术感。”



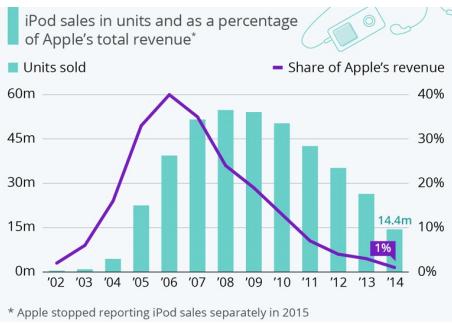
乔布斯也非常注重产品包装，iPod 的包装让你觉得：“在拆包装时，觉得有一件很贵重的东西在里面”。



虽然他定价不低：399 美金，很多人觉得这个定价当时是过高的。但是 iPod 在 1990 明年推出市场后一直热卖。

=====

从 2001 年到 2014 年苹果大概卖出了 4 亿件 iPod，2015 年后还继续卖出 50,000,000 件。iPod 播放器也成为了苹果的主要收入来源。在苹果推出 iPhone 前一年，2006 年，销售占公司总收入的 40%，详见下图：



## iTunes 网上商店

苹果的 iPod 加上 Mac 机上面的 iTune 很成功，但只解决了可以从网上下载音乐在手提播放器听歌的技术问题，但有更严重的问题要解决 – 盗版，2000 年开始，越来越多网站提供免费音乐下载，大大影响了唱片销售（2002 年销售额就跌了 9%），也侵犯了音乐歌曲创作者的知识产权保障，各唱片公司都继续解决这打击。乔布斯也很看重这些创作人的利益，他可以预见到如果没有知识产权的保障，就没有人再有动力录制高质量的歌曲和作品（如果软件与电脑没有知识产权保障，就没有公司愿意投资做创新）所以他就开始从办一个网上的统一的可以买电子，在网上购买歌曲的这种服务。除了技术方面格式问题要解决，很多有自己的格式，也不小要统一，还有各个唱片上都想保护自己的利益。比如有索尼，有划拉好几家大型的唱片公司，最终经过乔布斯的协调和平台的创建，他最终就建了一个我们现在看到的 iTunes store，都可以在网上搜索任何一个唱片公司的唱片或者歌曲，关键字搜索都可以，也有相关的唱片可以在搜索，要这种成功的话，乔布斯的概念很简单。他从以前的经验知道首先要廉价，让大批的人群进入使用这个网。如果收费太高，只是小部分人用，就流行不起来，也没有动力使唱片公司加入。他这个策略很成功。其实所以我现在也是每个月交几十块钱就可以随意随时在网上下载要听的歌，也从这个渠道也保护了艺术家、唱片公司的利益。

2023 年 4 月份苹果发布 iTunes 网上商店，开始时有 200,000 歌曲可以下载购买。本来预测这新服务可在六个月内销售 100 万首歌曲，但实际上推出六天就已经卖了 100 万，2023 年 12 月累积卖出二千五百万首歌曲；2024 年 7 月份共卖出一亿首歌曲（23 年底的 4 倍）；2005 年 11 月（两年半后），苹果成为全美国十大音乐零售商之一，其他都是传统零售商，例如，第一是 Walmart。随着网上买歌商业模式越来越普遍，传统零售商越来越困难，到了 2010 年 2 月，苹果成为美国最大的音乐零售商。

## 公司如何创新

从以上乔布斯关于音乐的故事，可以看到跟 400 年前不一样，400 年前纳皮尔的创新然后要 20 年，速度很慢，靠一个人独创。帮了人类进了一大步。到了 2000 年什么速度都变快了，乔布斯团队经过 3 个月就推出 iPod 播放器，也同时用了两年时间，把 iTunes 网上商店变成听音乐的一个公用平台，保护了艺术家、唱片公司，设备者各方面的利益，也帮苹果公司定位成一个数字中心，消费者

用他的电脑可以选择在播放器选自己的歌，也可以下载最新的歌曲，非常方便。这些就是我们 2000 年代看到的创新，这种的话绝不能靠一个人独自完成，而是要团队合作。

公司创新与几百年前那些大师的创新最大的区分是需要整个公司所有人的协作，大家目标也要一致。整个系统都要都要配合，但是不同人负责执行不同的工作，所以目标要细分。

纳皮尔先生用了 20 年，写了 3 本数学书（一本是他死后出版），总共不到 200 页。贝多芬生产率也不高，一年出版几首作品，这些大师都是靠自己一人完成整个创作。但是，现代市场变化越来越快，要成功，公司必须快人一步，但因依赖团队合作（各有专长），也需要协调大家的工作，确保各个层次的目标与公司总目标一致。

举例介绍企业如何用一个简单系统来管理整个公司的提升：

#### 1. 识别公司的总目标

例如在下一年，要一年内生产 7 个新的产品系统。

#### 2. 描述公司的现状

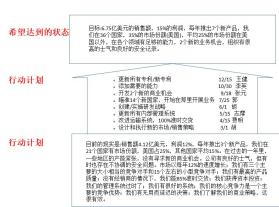
例如现状是：平均一年只能产出 3 个新产品，有 2 个开发团队。但开发人员的能力有限，质量也不好，导致用了不少精力，去维护已经完成的产品，没有时间来开发新的，招新人也不容易，比较难找得合适的，也没有时间培养新人，扩大工作。

#### 3. 形成具体行动计划

例如简洁、清减的开发流程，增加 1 个开发团队，加强代码评审和编码能力，来减少缺陷导致的返工，要跟市场部密切合作，确保新产品满足客户的要求，给客户带来价值。

但真正公司的从上而下总计划会更具体。

#### 4. 当我们定好明确的最终结果，也了解真正的现状，有一些行动计划，下一步是具体到每活动的完成日期和负责人。



同意可以用以上模式帮助公司创新吗？请看看乔布斯另外两个公司创新故事（非苹果公司），你可能有不同想法：

### 万事起头难

乔布斯 1985 年 9 月被迫离开苹果公司后，自己投资 7 百万美金，独力开创一家专门针对高校工作站市场的新公司 Next。乔布斯开始时对公司的期望特别高，当时他看到这市场主要被太阳（SUN）和（DEC）公司占有，但这些工作站产品都是以输入命令为主，不同于 Macintosh 用鼠标操作不同，所以易用性

不高，所以他希望创造一个前所未有的新平台，让那些高校的学者与研究人员可以简单地使用电脑，模拟各种场景，做研究。

但是要达到这么高的理想，要做的工作就非常多。比如第一次公司开创 90 天后就开始有一次度假村的头脑风暴会议，希望可以在两年后推出产品，赶上学校暑假购买的时期，再过了三个月，再讨论时，大家都发现这个要求太高，难以实现。但乔布斯没有放弃，他一直用自己的魅力说我们可以按大家的专长达到这个目标。

1986 年过去，1987 年又过去了，一直都没有把新产品研发出来。到了 1988 年开始做了第一次三个小时的产品演示，但产品还是很初级，例如都没有颜色，速度也比较慢，虽然签了以前为苹果推销 Mac 的全国电脑零售商，但一年才卖了 360 台，离目标差很远。（1984 年，那家全国零售商一年就为苹果卖了 40 万部 Mac；NeXT 工厂生产线是按每月生产 1000 台做设计规范！）

1990 年，第二次产品发布，新产品开始有了颜色，速度也提升了。但是还是一个初级的产品，未成熟。

因销售不理想，乔布斯再增加了 500 万投资，很快也用光。还好有一位外部投资者，他非常相信乔布斯的魅力（他觉得以前没有来得及投资他的苹果，一直耿耿于怀），他就非常愿意就投了两千万美元进 NeXT。

到了 1993 年开始裁员。从开始本来公司有 600 人，开始只是裁几十人，后面再顶不住然后裁 200 多人，接近公司的一半人数，后面又裁 200 多人，最终把整个硬件部分卖掉给日本佳能公司，只保留操作系统部分 NextStep。

## 从快破产变为投资回报最好的投资

乔布斯在 1986 年用一千万美金投资 PIXAR（“公司”本来是 LucasFilm 下属 30 多人的电脑部，由“星球大战”导演 George LUCAS 投资创立，主要支撑电影的动漫效果需求；因为乔布斯一直对美术很有兴趣，他经 Alan KAY 介绍去公司看展示便被这公司的产品深深吸引，觉得很领先，远远超前同行）：乔布斯开始时只是投资者，没有太干预，公司的软硬件产品，针对高端动漫专业人士使用，但他雄心勃勃，觉得产品很有特长，希望和苹果 Mac 一样大量，买给做动漫人士使用，做 3D 设计。例如，产品虽然功能很强，但都需要复杂的命令操作，不适合一般动漫人士使用（但其他如 Adobe 公司的软件，虽然功能没有这么强，但很容易用），便投入研发，希望改善不足；为了推销全国，他还开始开零售商店。乔布斯对公司的投入投资也越来越大，但产品销售一直没有起色，公司开始缺乏资金，1988 年，他开始紧缩开支，和裁员。直到 1991 年，与迪士尼合作 TOY STORY，使 PIXAR 起死回生，乔布斯共已投入了 5 千万美元（占他从苹果公司股票卖出股票收入的一半）。也因与迪士尼成功合作，让 PIXAR 能在 1995 年（与 TOY STORY 首次公演同年）挂牌上市，为公司取得 15 亿美金投资。PIXAR 与迪士尼继续合作，推出 5 部电脑制作动画，都非常卖座，于 2006 年，迪士尼为了防止 PIXAR 与其它公司合作，用 74 亿美金收购 PIXAR，也使乔布斯成了迪士尼的最大独立股东。（关于 PIXAR TOY STORY 制作故事，请看附件）

后面事实证明乔布斯以为一般动漫人士会喜欢用 PIXAR 的 3D 模型软件是错误的，只是梦想。但他的另一个希望：如何结合电脑数字技术和艺术，做创新确能梦想成真。PIXAR 与迪士尼合作制作新一代动漫片，从 1995 年的 TOY STORY 开始，把动画片技术提升一个台阶。

NeXT：本来希望大量生产高校用的工作站电脑，与太阳 SUN DEC 竞争，最终经过多年的努力，只卖出几百台，为了生存裁员和把硬件部分全部卖给佳能公司，只留下操作系统 NEXTSTEP。但因为这操作系统被苹果看中，NeXT 公司最终于 1996 年被苹果公司用 43 千万美金收购，也让乔布斯重返苹果公司（开始时当顾问），经过 15 年努力，使苹果公司再创辉煌。

从上面两个故事看到企业创新都会经历原本预料以外的环境。所以任何创业公司因为市场是未知，有超过 90% 的创新产品都可能以失败告终，所以无法用以上传统的企业策划方式。也验证了马云先生的名言“今天很残酷，明天更残酷，后天会很美好，但绝大多数人都死在明天晚上”。

## 公司创新成功要素

回看 iTunes 电子商店的和之前 iPod 的成功故事，和乔布斯之前的 NeXT 和 Pixar 故事，可总结以下成功要素/注意事项：

灵活、敏捷

从物种进化历史可以看到物种可持续的原则：

- 最强壮/巨大的物种也会灭绝（如猛犸象）
- 最凶猛的物种也会面临灭绝（如狮子老虎）
- 只有最能快速适应环境变化的物种才能长期传承下去

从 60 年代开始，商业环境快速变化，也只有能快速适应环境的公司，不断求变，才能持久，成为百年老店。

从乔布斯的 NeXT 和 Pixar 故事看到，要创建一些新的产品服务变数很多，失败机会很大。所以要随时变招适应变化。如果发现本来的想法、计划没有成果，就立马要改变。

这道理适用于很多科技公司，比如谷歌也有很多失败的项目，谷歌的宗旨是‘Fail Fast’，项目如果过了半年，九个月。没能达到一定的客户量，就会立马叫停，终止这个项目。很多科技公司都是年底，才综合考虑某业务是否持续，可能已经太慢了。

这也是敏捷开发的原则：不花精力于长远的策划中。因为需求常常变化，分成小迭代，每迭代冲刺，基于有限资源，尽力把每次迭代做好：不仅仅效率高，产品质量也要好。

## 专注 (Focus)

资源有限，只关注公司自己能做好的东西，哪些可以外包，就尽量外包。比如生产，苹果只专注做好自己的专长 - 产品设计。（请不要误会苹果不管生产，它的产品生产都是按 JIT 做，只是非直接聘用生产工人。）

乔布斯回到苹果后，吸收了过去十年在 NeXT 和 PIXAR 的经验教训，本来苹果公司里面一百多个研发项目，乔布斯很了解资源有限，便开始整合，只留下每个市场最好的研发项目，解散不需要的研发团队，只留下跟这个愿景相关的研发。

乔布斯 2010 年被访问时说：“我们选技术，只挑选有发展潜力的技术，保全宝贵资源，因什么都支持，都研究就会耗费宝贵资源。”

### A 级团队成员

Pixar 公司虽然小，但每位都是行业精英。

我美国堂弟从小很喜欢动画人物，家里全都是星际旅行、星球大战等模型。他来香港旅游也专门去那些小的专门店找他喜欢的卡通人物模型。他读建筑专业毕业后专攻动画片和主题乐园设计。他和其他 3 位设计师是公司外聘参加“超人总动员”制作设计的原始核心团队。其他 3 位都是行业中的有名高手，例如，Blackman 曾设计蝙蝠侠系列，Delgado 设计星际旅行 (Star Trek) 等。他说 “Pixar 公司在动画片行业非常出名。我们四位核心制作团队，包括艺术总监和生产设计师等都是当时行业的顶尖高手，他们除了能力超强，对艺术创作的品味与兴趣跟我也很相似，能有这机会与他们共事合作真是非常幸运，像中了彩票一样。开始时，我们四位设计主要成员定期要跟导演 Brad Bird 开会，评审设计。” “超人总动员”非常卖座，美国本土票房收入已经超过 2 亿美元，也得了奥斯卡奖。

Pixar 公司这些动画片，能有这么好的票房，并得奖，不是没有原因的，因为公司有名气，可以吸引到当时最优秀的艺术家参加，有了这些人，我们才可以看到像“超人总动员”的高端制作。

所以当 A 级成员遇上能力相当的团队伙伴，更能擦出火花，1+1 大于 2 的效果。

苹果公司 1984 年发布的 Macintosh，产品的技术和设计（例如，视窗，鼠标等）都远远超越了之前 IBM 出的 IBM PC。开发团队成员的能力很超强，其中包括 Burrell Smith 设计硬件线路版，Andy Hertzfeld 等设计 Mac GUI 操作系统，Bill Atkinson 设计 Graphics 应用软件 QuickDraw, HyperCard, Joanna Hoffman 负责市场推广，Bruce Horn 设计 Resource Manager 应用软件等。但产品发布后，因为销售不太好，管理层也换了人，团队都陆续离开了，

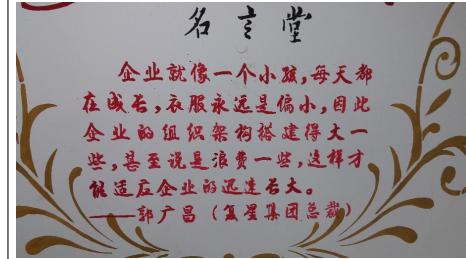
乔布斯从 Pixar 的经验，深明高端球员 (A Players) 只愿意与高端球员搭档的道理。但为了吸引和留住这些精英，公司必须给有吸引力的报酬。

苹果只招聘一流员工，所以团队能力非常强。无论在广告推销，软件开发产品样式设计，跟合作商谈判等等。

例如，乔回苹果不久就要求董事会修订员工股权安排（因当时苹果股价已经很低，之前的员工股权安排已经没有意义。）

极高目标，要做大做强

在杭州某商务酒店看到十几条浙商名言，写在墙上，其中一条如下：



乔布斯创立 NeXT 时是世界第一的工作站供应商，所以生产线是按每月能生产一千台设计！他重回苹果后，也一直按做出世界第一产品推动团队，后面事实证明，例如 iPod / iTunes(2001), iPhone(2007), iPad(2010)，这思路使苹果公司后面成为美国市值最大的科技公司。

丰田大野耐一说过，“容易达成的目标不是好目标”，他对丰田员工设的质量目标是零缺陷！

有非常高的要求，用心做

每位团队成员都有高度的要求，不仅仅是做完事，要做出优秀的、世界一流的产品，有这种爱心，才可以驱动每一个人辛辛苦苦付出最大努力把这件事做好。例如乔布斯自己就是一个典型例子，他信心这些是有价值所就全心去做好。比如在推出 iTunes 电子商店以前他就不断的邀请各种音乐家、歌手、音去家。他每次都会想尽办法去介绍他的好东西，有一次小号手 W. Marsalis 到家拜访，乔布斯就问他：“你喜欢什么歌曲？”

Marsalis 说：“贝多芬”，他就一起展示如何在商店里面可以容易搜索到，Marsalis 后面回顾说：“其实我一直都不太对电脑感兴趣，但乔布斯他一直全心全意，全程用了两个小时很细心地给我介绍。过了一会，我的注意力不是放在他的产品或者电脑上，而是他本人，我深深被他的激情/专注打动。”

例如苹果的首席设计师 Jony IVE 用简洁设计思路，让消费者觉得苹果产品有艺术感，非一般电脑：



Ive 从他父亲就学到要做一流工匠必须对质量有极高要求，。。。

#### 领导决策、快速行动

前面不是说索尼也一直在电子产品很创新，也有自己的百人音乐公司，有唱片公司。全部都有他的能力，但是为什么他做不出来，还是让苹果成功。其中一个最大原因是索尼已经是一个大公司，有各个事业部门组成，每个事业部要管自己的盈利，也是真正这个原因，要这种团队之间协作的，预计就做不出来了，很多精力就耗在内耗，内部部门之间之间的协商。反过来，苹果是不会这样算的，他只是算整个公司的毛利，不会在细分每个事业部自己，而且乔布斯有非常高的要求，

例如，整个 iPod 时候的那个设计，当工程部和软件开发部做了那些原型初稿 30 个，然后乔布斯进来以后大家都同意这个款式，就基本上立马行动去做了，但是有些在比如飞利浦做过的人会觉得很奇怪，那如果像西门子、飞利浦那种传统大公司绝对不会这么重大的一个投资设计，会在一个会议里面搞定。

#### 结束语

苹果的例子可以看成是敏捷开发升级到公司级的最佳实践案例实例，从前面对的故事看到整个公司都是按精益的思路去做。

团队合作性很重要，团队的能力也应该要很高，才可以在最短的时间做到最好。也强调岗位之间的互相合作，不是你是做测试，我是做开发，你是做需求我只管做好自己的工作，大家相互合作，最终希望整个项目产品成功，不计较是否在我的工作范围之内，也确保相互是打通的，你自己需求或者开发做的最优，不表示最后产品可以做到最优，有些开发团队抱怨：“我其实开发挺不错的，但是需求一塌糊涂，我们无法用好。但是也因为公司架构原因，部门之间各有自己的领导，自己的指标，导致功能之间不能相互合作做好，“

从乔布斯故事了解高层有高度的要求很重要，要做好敏捷开发的一样。如果敏捷团队不注重产品质量，只关注按时交付，最后只能做出一些烂产品；但反过来，当团队有能力，大家有抱负有要求，敏捷开发可以帮助团队快速反应不断变化的需求，像苹果与乔布斯一样成功，世界第一。

1983 - 84 年，乔布斯（28 岁）在苹果公司带领 Mackintosh 开发团队时，就不会向预计交付期限低头。如果他觉得产品未达到质量要求，他宁愿延后。他甚至鼓励团员要精益求精。（千万不要以为在他管理的团队可以偷懒）

有人问我：听过 SAFe 这个框架吗？我们想用它在公司级推敏捷，你觉得怎么样？我：框架只是框架，如果你没有我们刚才说那些要素，什么框架都帮不了你。因为那些框架只是把一些最佳实践的要素列出来，有了那些过程，有了一些度量，没有合适的人，没有合适的领导，没有苹果这种公司文化，永远不会做到世界一流的产品。这道理国内外同样适用。

====

2005 年，乔布斯获史丹福大学颁授荣誉学位，他在典礼中跟其他毕业生讲了三个故事：

故事一：

高中毕业后，十七岁，我进了私立的里德大学，学费昂贵。读了半年，我一方面觉得学非所用，另一方面不忍心花掉父母一辈子的积蓄，就退了学。但我并没有离开学校，而继续在学校里旁听感兴趣的课。我没有收入，就睡在同学宿舍地板上，同时靠捡玻璃瓶、可乐罐挣点小钱糊口。因平日吃不饱，每个星期天，我走路到距离七英里的一所印度寺庙去吃一顿施舍饭。

大学的美术字 (Calligraphy) 课程很有名，我去旁听，立马迷上了。虽然当时我还不知道以后有什么用，但是后来在设计苹果的麦金托什 (Macintosh) 计算机时，想到当年在大学里旁听的课程，为 Macintosh 个人电脑设计了很漂亮的字体。

十年后回看，这些生命中的小点都能连起来，但不可能之前预计会连起来，只能事后回顾能看到。所以我们必须相信这些小点会在生命中某时候能连接起来，才有信心按自己信念勇往直前。

故事二：

我在 30 岁的时候被苹果公司赶了出来，觉得很失败，没有面子，对不起那些其他创业者。后面因为我还是对电脑 IT 很有兴趣，我就自己开公司继续做产品研发专门针对工作站希望开发一些世界一流的产品，我现在回想看如果当时没有离开苹果，继续在苹果的话，后面就体验出很多我产品的开发的概念。所以我现在回想过来，一点都没有后悔。最重要就是你必须要热情的爱上，无论是工作或者找伴侣都是一样道理。

“有些时候，生活会拿起一块砖头向你的脑袋上猛拍一下，不要因此失去信仰。我很清楚，支撑我一路走下去的，是那些我所爱的东西。你需要找到你的所爱，工作如此，爱人也是如此。你的工作将会占据生活中很大的一部分。你要相信这份工作是伟大的，你必须先热爱它；你只有坚信自己所做的是一份伟大的工作，才能怡然自得。如果你现在还没有找到，那么继续寻找，不要停下。只要全心全意地去寻找，在你找到的时候，你的心会告诉你的。”

### 故事三：

我十七岁时，听说应把每天当成你生命最后一天。所以我每天都会对着镜子问自己，回顾是否已经用好当天，做最重要的事情，如果连续几天都否定，就必须马上调整。但现在回想，如果保持这想法，能逼自己只做真正重要的事情。我会每过几天，如发现把时间浪费在非最重要的事，会立马调整过来。

“死亡是每个人必然的终点，没有人可以逃脱。死亡是生命中最好的发明，起新陈代谢作用。记住每个人都会在不久后死去，对我作用重大：当我做重要决策时，所有的顾虑：其他人的期待，怕失败没有面子等，相对死亡都会变成微不足道。

当你们知道人生时间有限，便再没有理由不去听从内心指引，专注做好最重要的事，因其他事情都不再重要。”

## 个人经验教训

乔布斯先生在大学毕业典礼上分享他的三个故事，回看我自己过去的人生旅程也有同类故事：

我预科考大学的成绩非常好，除了语文以外，物理化学课数学都全 A 进香港大学选读什么系都没有问题，我父亲自己没有读过大学，基于传统思维极力劝我读医科。（我祖父母的年代，香港大学可选择科目不多，学费也高，不是一般普通家庭可负担，所以父母都想极力子女读医，毕业后收入有保障，社会地位也高）我自己自问一直对生物、化学兴趣不大，反而对数学从小都一直很感兴趣，（本来一直想读麻省理工，或计算机类专业）但香港大学还没有计算机系，便选了的电子电机工程。但父亲很反对，甚至威胁说如果我不选读医便跳海！我前思后想，最终还是不变，决定选读电工。

因为香港工业不发达也不注重科技，电子工程毕业在香港没有好出路；读书时和毕业后的 10 多年，还会时不时怀疑是否如当年听父亲话会更好。

现在过来还是非常正确的选择，如果选了医科便没有机会 5 年前自己再读线性代数，统计分析，大数据等我很感兴趣的课题，也无法在过去 10 多年一直在大陆各地跑，长期出差。

1979 年在香港读预科时首次读《矩阵代数 (Matrix Algebra)》，觉得很新奇，但不知道有什么应用。（但牛顿力学就实际多了，例如可以用来计算台球碰撞后的轨迹和速度也可以用来计算行星和月球的轨迹。）

1998 年在香港富士通工作时开始接触到软件工程，觉得很多不懂，但也很有兴趣的兼读软件工程硕士，开始学敏捷开发、面向对象软件、易用性、软件质量等。大学毕业后晚上兼读管理文凭时首次接触到 XY 理论，2010 年读 MIT 教授 McGregor 的经典书 “The Human Side of Enterprise”，开始接触和组织开发 (Organization Development) 的各种实验和研究，也看心理学的公开课视频。

2017 年开始接触大数据分析，发现要弄懂大数据背后的原理，必须了解矩阵代数，矩阵代数的另一种名称，便开始在网上搜索相关视频。2018 年，从麻省理工公开课 (MIT OCW) 视频里找到由数学教授 Prof. STRANG 主讲的 18.065，一共 34 篇讲课，很欣赏他的讲课方式，用粉笔在黑板上写，加上讲解，一直有空便看视频。

预科读的书《矩阵代数》我一直保存，作者也是 Prof. STRANG，发现这本参考书依然很实用，里面的内容完全不需要更新，因数学，与工程或技术不同，原理不变。除了再读矩阵代数那本书，我也买了 Prof. STRANG 为这课程新出的书，经过几个月，看完接近共 30 课。

现在回想以上每一点都对我的咨询培训工作和写这本书有直接帮助。

====

我也是曾经被公司开除，在我 40 岁那年，被富士通开除，我当时是业务经理。开始时很迷茫，觉得自己很失败 – 从刚毕业时的大学精英，一下变成没有工作的失业汉。现在回看，如果当时不是在 40 岁从零开始估计会现在我会和其他同学一样，50 多岁就被迫退休，也不可能再找到工作。

====

大概 10 年前左右，看完 “最后十四堂星期二的课”话剧也读了原文小说 “Tuesdays with MORRIE” 在与家人去爱尔兰度假时，开始想到生命有限，人无法控制命运安排，人随时可能死亡，必须抓紧时间，尽量把想写的东西记下了。（COVEY 先生经典书 “Seven Habits” 中习惯二 (Begin with the End in Mind, Principles of Personal Leadership) 的场景：你参加自己的葬礼，看到你家人在参加葬礼的亲友前读出自己的一生。）

我便开始录音，写分享文章。开始时，只能试短的分享文章，语文差，写得很慢很费劲，读多写多逐渐完善，最终出版自己第一本书。（也对应乔布斯的故事：只有事后才知道原本选择的努力有没有用，原先只能靠直觉、兴趣。）每人一生成就不是看是否名牌大学毕业，而是取决于能否找到最爱，并付出努力，全身全力做到最好，并且定期回顾。把每一天当成是人生最后一天，只干最重要的事。

“不积跬步，无以至千里；不积小流，无以成江海” —— 荀子《劝学》

## 附件

### Pixar 制作第一部电脑动漫：玩具总动员 (Toy Story)



John Lasseter 出生于西岸 Hollywood，从小就非常喜欢卡通片，从加州美术学校（迪士尼出资创建）毕业后就进了迪士尼制作动画。他与其他刚毕业的动画师一样，都很想创造一些像当代星球大战那种高质量作品，但部门主管都非常保守，按传统方式管理，经常发生冲突，后面他被公司开除。刚好， Pixar 想自己制作短片，展示公司的软硬件技术能力，便招聘了他（动画师都经过多年专业培训，收入不低，为了避免管理层质疑，Lasseter 入职时职称是“界面设计工程师”）。

因为乔布斯很喜欢艺术，而 Lasseter 是公司里唯一懂艺术的人，所以从收购公司开始两人合作一直非常默契。

1986 年，公司决策要制作 2 分钟动画片展示公司的最新 3D 技术。

他就按自己所长，按他天天相对最熟识的桌灯制作动画短片。

同行看完短片初稿后反馈说，“虽然是 2 分钟短片也必须讲故事”

“短短 2 分钟，怎么可能讲故事？”Lasseter 想，但他还是按这建议，改成做成两把桌灯，一大一小：大的是父亲，小的是小孩。相互争夺一个气球，争来争去，最终气球被弄破了。（大家可能都看过，因后面迪士尼 Pixar 制作的动画片都会在正片前播放这短片。）

乔布斯，虽然 NeXT 的工作非常忙，但还抽出时间与 Lasseter 一起参加 1986 年八月份的动画片大会 (SIGGRAPH)，短片非常受欢迎，并获得大会的最佳动画片奖。

乔布斯时后领会到： Pixar 公司唯一这些短片才真正含艺术元素，不仅仅是科技技术。结合艺术与技术用于电影制作是公司的唯一出路，与他 10 年前在苹果公司结合技术和艺术创作 Mackintosh 一样道理

1988 年，虽然公司资金非常短缺，一直靠乔布斯不断注资，他还给 Lasseter 动画片部 30 万美元制作短片，但叮嘱必须做出优秀作品。不负所望，‘Tin Toy’得了 1988 年短片奥斯卡（学院奖），迪士尼高层想招聘 Lasseter 制作电影，但他不愿意离开 Pixar。迪斯尼与 Pixar 开始谈制作玩具总动员 (Toy Story) 动画片。虽然乔布斯是谈判高手但因为 Pixar 是小公司，急需有这种新项目维持，经过几个月的商务谈判，双方签了商务合同，因迪斯尼有绝对强势， Pixar 只是迪斯尼的合同工：迪斯尼拥有所有里面的人物和电影片的版权，后面从票房销售 12.5% 给 Pixar，迪斯尼有权利继续用这个题目和 Pixar 合作两套电影，但迪斯尼可选，也可以随时停掉，不需要给任何赔偿。

从 1991 年开始就开始制作，但因为迪斯尼的负责人把里面的主角吴迪 (Woody) 变成一个很有性格单没有人性的牛仔，导致团队辛辛苦苦做出来的初稿在 1993 年 11 月被迪士尼高层叫停。但乔布斯还是没有解散团队，觉得还是应该有希望继续，团队按 Pixar 团队本来的思路改善动画片，三个月后再给迪士尼高层看，重获迪士尼的高层继续支持。1994 年二月份他们同意重新制作，但因已经发生了大量返工，本来的 1700 万美金预算无法完成，乔布斯觉得这些超支应该由迪斯尼承担（因这些超支都是迪斯尼管动画片主管的失误措施引起），但迪斯尼主管不同意。最后双方多轮谈判协商，最终调整了预算。乔布斯本来没有太多参与玩具总动员 (Toy Story) 的制作，但后来他越来越投入，到后期，初稿出来后，他每次都会把更新后的动画片，邀请亲友们到他家里看，有亲友回顾说：“每次去乔布斯家都要看他展示动画片的最新版本。虽然可能只是优化了不到 10%，越来越觉得有点烦，”

1995 年乔布斯被迪士尼邀请一些大型的动画片宣传活动，他预计到可以依赖动画片制作让 Pixar 公司获得投资资金，所以他同时筹备在 Toy Story 开片同年把 Pixar 公司上市（做 IPO）。但很多投资公司怀疑会否成功，原因是 Pixar 公司过去五年都是亏损的。乔布斯回顾：“当时确实有些紧张。也有人劝告我应该等到第二部电影出来才 IPO。”但乔布斯解释说：“我们急需这些钱，让我们跟迪士尼可以谈判第二部电影，如果我们没钱，无法谈判更好的合同条款，我们永远只是迪斯尼手下的合同工，没有什么发言权。”玩具总动员异常成功，第一周就有三千万票房，已经超越所有制作成本，然后一直成为当年票房最高的电影，在美国本土有 192 百万美金，国外有 362 百万美金，电影评价也异常好。

## 参考 References

1. Robert FRITZ, 'The path of least resistance for Managers'
2. Walter ISAACSON, 'Steve Jobs'

# Chapter 26

## 根与翼

中国社会一直非常强调家庭价值观，希望实现家族的持续传承（如：四世同堂），代代相传的关系对每个家庭成员的成长产生深远影响。我们每个人都只是人类进化过程中的短暂过渡。父母普遍希望把最好的东西传承给下一代。然而，我们需要问自己，什么东西能够持久，仅仅是财富和金钱吗？

继承大量财富，但如果不懂得如何正确利用，有时可能反而会削弱个人的竞争能力，并可能导致不负责任的行为，最终成为社会的蛀虫。

“七个习惯”的作者史蒂芬·柯维（Stephen Covey）在书的结尾（最后一章）中提到，我们可以传承给下一代并具有持久价值的，只有两样东西：根和翼。

'There are only two lasting bequests we can give our children - one is roots, the other wings.'

### 根

回看人类过去一百三十年，除了打了两次世界大战外，也做了很多前所未有的创新：

- 电灯（煤气灯）
- 汽车（马车）
- 飞机
- 摩天大厦
- 半导体，集成电路
- 计算机
- 互联网
- 移动电话

人类创新并不仅限于前述领域，这只是我们看到的凤毛麟角，其他在医疗、电影、音乐、数学等各个领域都表现出了卓越的创新力，这些都是祖先留给我们的宝贵财富。

举例来说，如果十九世纪欧洲城市的居民能够坐上像“Back to the Future”那样的时间跑车，见到两百年后现代人的生活，他们将感到难以置信。

然而，当我们观察现代社会的儿童和年轻人时，由于不再需要担心衣食住行，他们往往将更多的精力投入到虚拟世界的游戏、养宠物、以及在线购物等方面，这确实引发了一些担忧。

有人可能会提出反对意见，说：“我不是贝多芬，也不是乔布斯，我没有像他们那样非凡的魄力。”

### 东北虎

中国东北虎作为濒危物种，目前在国内的数量估计不超过 30 只，因此国家动物保护组织采取了多项措施予以保护，其中包括人工饲养。专家曾试图将这些人工饲养的东北虎重新放归野外，但这一任务极为艰巨。它们是否能够在野外独立存活下来，仍然是一个未知数。有一部纪录片生动地讲述了动物保护组织在这一过程中所面临的各种挑战和困难。

我们经过严格的筛选，挑选出最具潜力的老虎。第一次选中的是一只年轻且体型较大的老虎，将其放到一个模拟自然环境的大型操场，供专家进行观察。刚开始，这只老虎还是表现出了典型野生老虎的特征，如：警惕地观察周围的环境等。然而，几小时后，这只老虎没有抵抗住低温环境，还是选择回到自己的巢穴，这次尝试失败了。另一只被选中的老虎虽然充满活力，也不怕冷，但却缺乏野生动物的警惕性，它已经习惯了人类的存在，这可能会给它的野外生活带来危险。因此，专家不得不放弃了这只老虎。

要确保老虎在野外的独立生存，它们必须能够自行觅食。在自然环境中，它们主要的捕猎对象就是鹿，但鹿非常敏捷，时速可达五十公里，而且具备很强的耐力。因此，专家需要评估老虎是否能够成功捕猎到鹿，否则就不能将它们放归到野外。在评估过程中，专家发现老虎的奔跑和狩猎能力明显不如野生老虎，因为缺少运动，它们缺乏狩猎的能力。因此，专家设计了多种锻炼方法，如使用车辆牵引轮胎，并插上鲜肉块来刺激它们奔跑。经过一系列的锻炼，老虎的体力得到了恢复，但能否成功捕猎到鹿仍然是未知数。

另一个试验是用汽车拖着鹿的尸体，以此吸引老虎奔跑。然而，有些老虎因为在平时的饲养过程中并没有吃过鹿肉，所以对此并不感兴趣。最终，只有两三只老虎追着汽车跑，试图抓到鹿。最后，有两只年轻的老虎获胜了。但要将鹿的尸体作为奖励喂给获胜的老虎也不容易，发现它们对着鹿的尸体不知道如何下口，因为以前都是由饲养员将肉切好后喂给它们吃。最终，这两只老虎花了数小时才慢慢学会如何吃到鹿肉。

看完这部纪录片，我立刻想到了有些年轻的程序员，他们也是在相对受到保护的环境下工作。这些程序员可能已习惯了管理层或经验丰富的同事对他们的代码质量要求并不高。因此，如果我们希望这些年轻的程序员注重代码质量，就跟保护区把饲养的东北虎放归野外一样，需要经过艰苦的培训，以提高他们的竞争力。管理层必须认识到，只是一味地保护程序员，对他们提供高质量的代码没有要求，那么公司的产品质量将永远无法提高，也难以与其他公司竞争。

我曾与一位企业质量官进行交流，发现越来越多的企业开始注重质量，尤其是在客户对质量要求非常高的行业，如银行、基金和保险等。他说：“公司制定质量流程非常容易，但是很多公司在执行过程中出了问题。根据我近二十年的从业经验，企业文化起着关键作用，因为提供高质量产品的核心是个人都要具备质量意识，并将其养成习惯。

## 翼

除了根的传承，祖先还留给了我们“翅膀”，赋予了我们自由的能力，以此打破负基因的传承，追求改善、变革和创新，而不是简单地将这些基因传给下一代。

To be a powerful transition person, one must first change his inner first.

我们每个人都只是在人类几万年进化历程中的一个瞬间。必须先从内而外做改变，才可以持久。要成为一位有影响力的过渡者，我们必须首先进行由内而外的改变，这样的改变才能持久。

当然，我们也可以在自己可控的范围内进行创新，或至少有所改善。

## 飞行实验

人类一直都想飞上天，中西方对此都有尝试。例如，传说明代火箭专家万户陶成道尝试在凳子上绑 47 根火箭，又绑上风筝做飞行试验，结果爆炸而亡。15 世纪，意大利人达芬奇也模仿鸟类的翅膀设计飞行器，但只是留在图纸设计阶段。

韦德兄弟 (Wright brothers) 1903 年首次机动飞机飞行成功莱特兄弟 1903 年首次成功飞行，在 12 月 17 日的第四次飞行在空中飞行了 59 秒，航程达到 259 米。下图是 1903 年滑翔机飞行照片：



### The Wright Brothers – First Flight in 1903 |

莱特兄弟并非工程或数学天才（他们都没有读过大学），但为了实现“飞翔”的梦想，不断从失败找改进，从 1899 年开始不断试验：

1900 年，他们首次试验滑翔机，但效果并不理想。

1901 年，他们把机翼的面积从 15 扩大到 26 平米，可以飞行 120 米，但他们发现试验数据与计算数据不符。为了找出原因，他们制造了风洞来测量各种机翼设计的浮力。他们试验了 200 种机翼。利用收集数据，他们开始制作第三架滑翔机。

1902 年 9 月份，他们完成了 1000 次试飞，其中空中飞行最长可以达到 26 秒。也因为这些经验，他们知道如何设计飞机的舵，让飞行员更好控制飞机，并开始设计使用 4 气缸发动机加上螺旋桨（设计原理和机翼设计相同）准备设计机动飞机。

1903 年成功飞行后，他们继续改善设计。1905 年 10 月，飞机能在空中飞行 39 分钟，并且能在飞行员控制下转圈。

在韦德兄弟之前，德国的飞行之父 Lilienthal 已经进行了大量的计算和研究，试图实现人类在空中飞行，但却没有成功。他们回顾说：“我们需要更多的技巧而不仅仅是机器。我们试图向鸟类学习成功飞行的技巧，而不是试图挑战大自然，完全参照鸟类，像它们一样。”

韦德兄弟根据 Lilienthal 先生的计算结果，在风洞里进行实验，发现他的计算存在错误。他们发现鸟类在飞行时会使用翅膀的尖端来控制方向。此外，他们发现为了平衡飞机在机体旋转而产生的扭矩，会使飞机向反方向转动，影响飞行。飞行是在不稳定的大气条件下操控不稳定的机器，充满了许多变数，因此非常困难。最终，韦德兄弟通过不断实验，自己发明了飞机各种重要部件，包括机翼、舵、发动机、螺旋桨等，并研究了它们之间是如何相互影响。

从莱特兄弟的飞行故事，可以看到创新不是单靠创新的意念，更重要是不断试验，从失败中获取经验教训，收集数据持续改进。

1985 年乔布斯被迫离开苹果公司，独资开创 NEXT 公司，到 1997 年 NEXT 被苹果公司收购，乔布斯重返苹果（当时苹果公司在破产边缘），到 2010 年将苹果公司带回到美国市值最高的科技公司，中间也经历了多次失败。

丰田公司从二战后快破产到后来成为世界第一，也非一帆风顺。

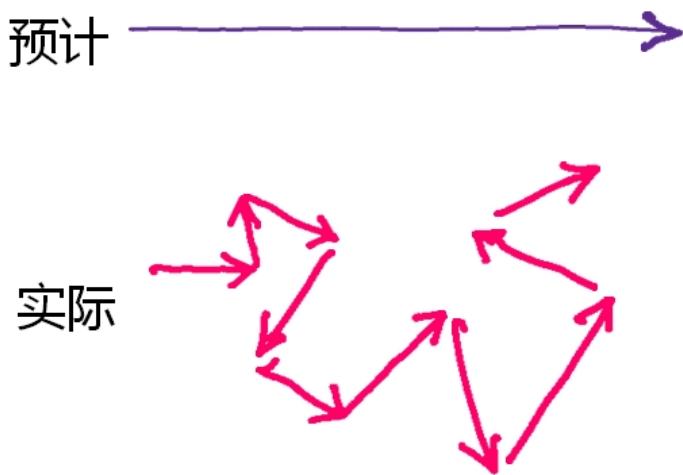
2023 年底，总收入可能是大众第一，丰田第二，但丰田销售数量世界第一。)

## 总结

针对软件开发，我们这一代继承了之前从 60 年代以来的各种硬件软件的创新，给予我们继续发展的根。

敏捷宣言指出了传统瀑布式开发和传统计划驱动开发的弊端，给我们以“翼”，为有能力的团队提供了机会，可以更快速地开发出高质量的产品。

提升质量和效率是一个逐步改进的过程。开发团队有了依据数据持续改善的动力只是第一步，更重要是在每次迭代回顾做实验，收集数据不断改进。



团队和企业要健康成长依赖以下主要因素（与上一章创新的要素类似）：

- 团队成员的能力与动力
  - 个人提升、自我管理、代码质量等部分能帮助提升
- 改进要有专注
  - 例如针对如何减少缺陷返工，尽早在评审和前期测试发现缺陷，如何做好迭代回顾部分能帮助团队了解应如何开始
- 高层要制定高的目标，也要提供资源
  - 可参考开始过程改进之旅里面的获取高层支持，尽早估算改进能节省多少成本，要求立项，把过程改进作为项目来管理

团队的持续改进至关重要。即使在未来遇到像前言中提到的那位严格的技术总监——陈总，也再不会被挑出一大堆毛病了。

相反，如果一家软件开发公司只注重项目的按时交付、追求急功近利，而忽视了产品质量，结果将导致不断出现问题和返工，最终会被那些持续改进的竞争对手抛在后头。

预祝你们公司会成为软件开发的丰田！

## **Part IX**

### **附录**

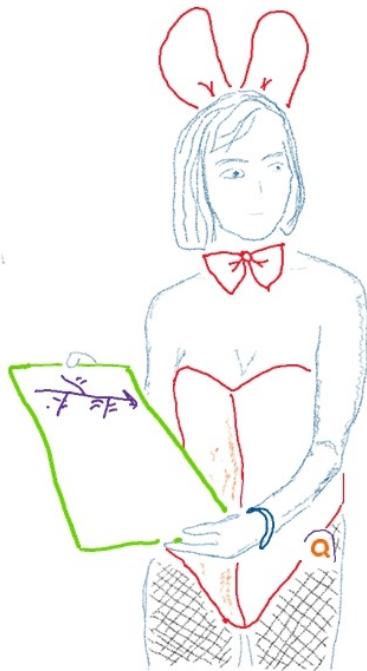


## Chapter 27

### A: 绅士俱乐部过程改进

#### 绅士俱乐部过程改进

美国学者 Dr Wheeler 到日本做培训，晚上好客的日本人请他去绅士俱乐部 (Esquire)。他发现年轻的女服务员腰上都有一个英文 Q 字牌子，便好奇地问原因。原来这家俱乐部刚完成几个月的过程改进，得到政府奖励金，所以都挂上这个 Q 牌 (Q 代表 Quality 质量)。



以下让我们看看这家俱乐部如何做过程改进。

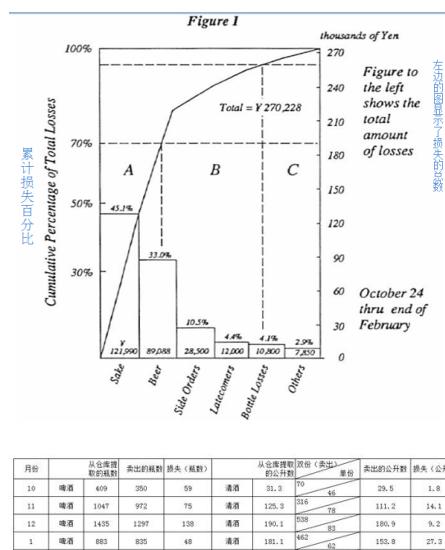
### Esquire QC 圈

QC 圈成立于 1984 年 12 月，由 Sakae 领导，下面的女服务员。

当时，Sakae 43 岁，有七个月的经验。除了她，圈内 7 女服务员的年龄从 19 岁到 23 岁不等。

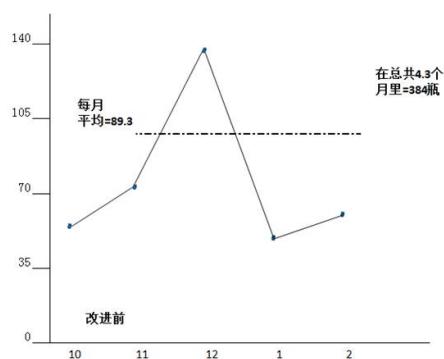
### 用二八原则（Pareto 图）识别主要源头

- 清酒 (Sake) 与啤酒 (Beer) 是最大的源头。

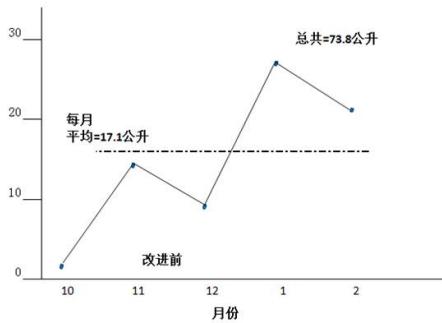


月份	从仓库领取的瓶数	卖出的瓶数	损失 (瓶数)	从仓库领取的总金额 (卖出)	卖出的总金额	损失 (公升)
10	409	350	59	酒酒 31.3	酒酒 29.5	1.8
11	1047	972	75	酒酒 125.3	酒酒 111.2	14.1
12	1455	1297	138	酒酒 150.1	酒酒 138.8	9.2
1	883	825	48	酒酒 101.1	酒酒 96.2	27.3
2	844	780	64	酒酒 151.0	酒酒 129.6	21.4

### 每月啤酒损失量



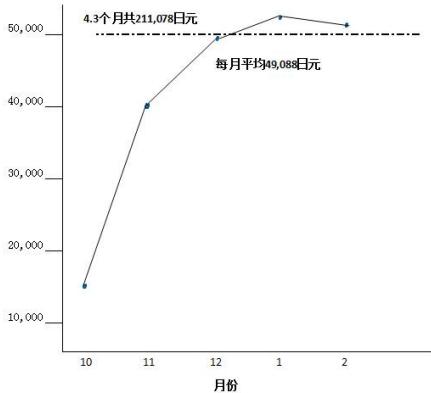
### 每月清酒损失量



结算出平均每瓶啤酒 232 日元，每公升清酒 1,653 日元

月份	损失 (日元)		每月损失	累积总数	支出 收入
	啤酒	清酒			
10	13,688	2,975	16,663		84.10%
11	17,400	23,307	40,707	57,370	85.40%
12	32,016	15,207	47,223	104,593	82.80%
1	11,136	45,127	56,263	160,856	77.60%
2	14,484	35,374	50,222	211,078	79.40%

啤酒和清酒每月总损失 (日元)



从 10 月份到 2 月份，啤酒和清酒的平均每月损失约 49,000 日元。

### 建立目标

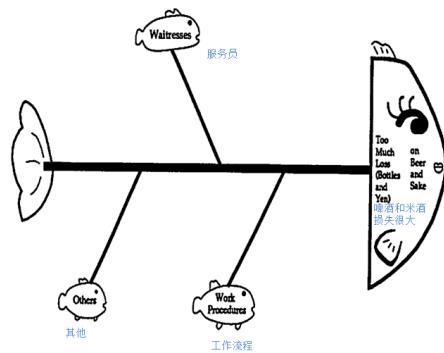
目标：把损失减半（少 50%）：从目前每月平均损失 89 瓶啤酒和 17 升清酒，降到（不超过）44 瓶啤酒和 8 升清酒。

### 根因分析 Root Cause analysis ( 鱼骨图分析 )

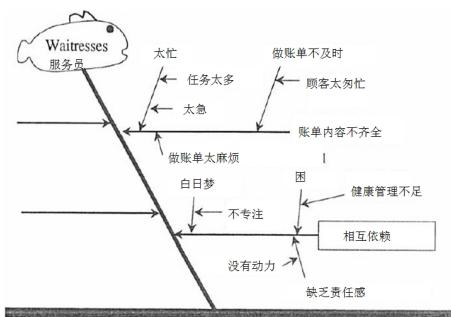
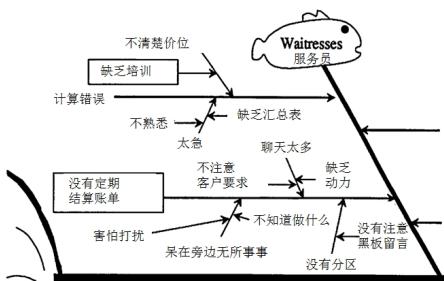
- 大家头脑风暴，讨论分析啤酒和清酒损失的主要原因。

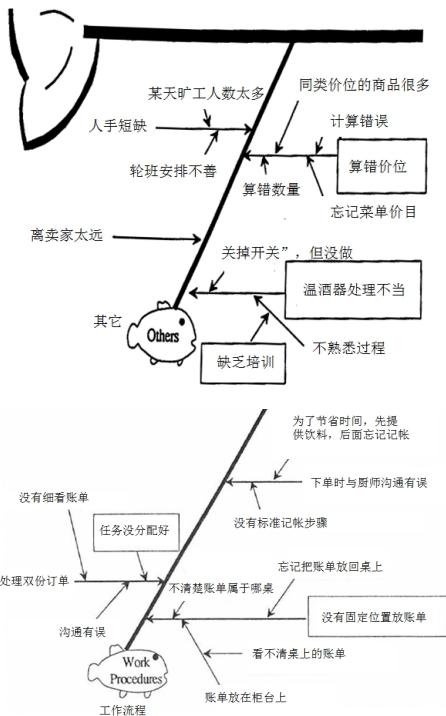
- 根据服务员、工作流程、其他等三个主线分析。

这三组构成因果图的三个主要分支。



鱼骨图每个分支的主要内容如下：





基于上面的鱼骨图分析，大家讨论并想到了以下主要理由/原因：

- 有具体的固定位置放置账单。
- 依赖别人。
- 处理热清酒器不当。
- 价格计算有误。
- 分工有误 / 没有分配好工作。
- 准备不足。
- 对客户桌结算（喝了多少并收拾空酒瓶）频率不足。

注意：以上每一项都是问题的根本原因，很多人误以为针对问题本身，找改正措施便算根因分析。“5 Why”方法可以帮助正确找到根因，详见附件“5 Why”例子。

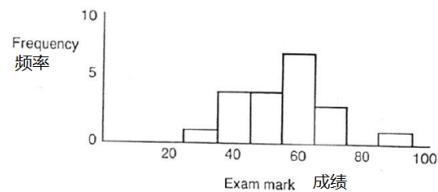
### 改进措施

- 要确保保留票据在桌子上，不带走（在繁忙时段，在纸上标记消费了多少矿泉水、啤酒和日本清酒，贴在总单的后面，利于计算总账）。
- 制定规程来分配各个区域的工作。每个小时每个区域负责人必须检查一次所有范围内的票据。

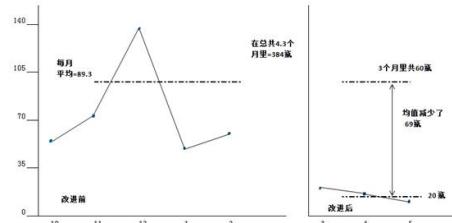
- 送热毛巾的那位服务员负责记录新加入的客人
- 谁接单谁负责记账，如果她要求其他人来协助记录，必须确保沟通好，检查是否已经做好记录。
- 管理者必须对新成员提供培训，如新员工小组学习。
- 以区域分桌，使大家清楚谁负责哪桌。

### 改进效果

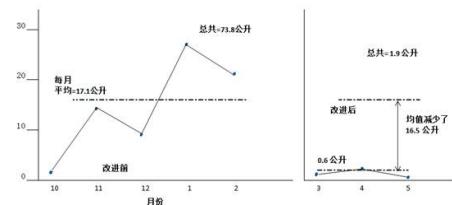
600px



### 改进后啤酒损失降低到平均每月 20 瓶

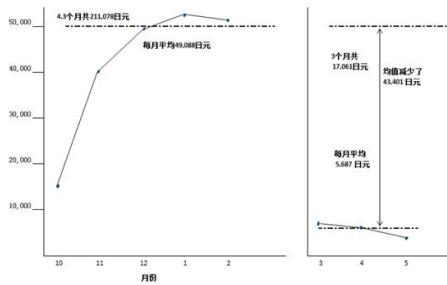


### 改进后清酒损失降低到平均每月 0.6 公升



结算出平均每瓶啤酒 232 日元，每公升清酒 1,653 日元

月份	损失(日元)		每月损失 累加总数	支出 收入
	啤酒	清酒		
10	13,688	2,975	16,663	84.1%
11	17,400	23,307	40,707	85.4%
12	32,016	15,207	47,223	82.8%
1	11,136	45,127	56,263	77.6%
2	14,484	35,374	50,222	79.4%
3	5,336	1,653	6,989	72.5%
4	8,472	1,157	6,029	74.1%
5	3,712	331	4,043	65.0%
			17,061	



以日元计算，啤酒和清酒的损失从每月平均 49,000 日元减少到 5687 日元，减少了 43,401 日元，即 88%。这一降幅远超过最初设定的 50% 的目标。

### A3 报告

很多给管理层的报告都非常厚，信息太多导致管理者难以消化，所以丰田要求所有管理报告都必须精简到一张 A3 纸。照片中服务员手上的就是 A3 报告 --- 包含了根因分析的重点。例如你可以看见其中有鱼骨图。

### 参考 References

1. Wheeler, Donald J. : SPC at the Esquire Club (SPC Press 1992)



# Chapter 28

## B: 分析迭代客户满意度调查数据

### 案例背景

香港公司在广州的离岸软件开发中心，专门为香港的客户，如政府部门，做软件维护工作。从 2019 年，项目已经开始采用 SCRUM 敏捷开发方式，每两周一个冲刺，他们每次做完迭代后，客户都会填写满意度调查，然后分析数据。

### 迭代数据

每次迭代团队都会收集以下数据：

- 客户满意度调查（Cust Sat survey）
- 需求变更请求次数（Requirements and Change Requests）
- 系统测试缺陷数（Test defects）

### 数据分析

- Overall 满意度（整体指标）与 7~8 个因素相关，而其中的 Deliverable Quality 相关系数达到 0.90，Deliverable Time 0.688 左右，其他的相关性都较低。

	P1A			
	Sprint 49,50	Sprint 51,52	Sprint 53,54	Sprint 55,56
	5	5	5	5
	5	5	5	5
	4.5	4.5	4.5	4.5
	4	4.5	4	4
	4.5	4.5	4.5	4.5
Deliverable time 交付时间	4	5	4	3.5
Deliverable quality 交付质量	3.5	4.5	4.5	4
Overall satisfaction 满意度	4	4.5	4.5	4

	P1A			
	Sprint49,50	Sprint51,52	Sprint53,54	Sprint55,56
Deliverable time交付时间	4	5	4	3.5
Deliverable quality交付质量	3.5	4.5	4.5	4
Overall satisfaction满意度	4	4.5	4.5	4

- 测试缺陷数和以下客户行为相关性如下：插入任务 0.51，需求模糊 0.66，初期没有需求问题 -0.45，需求不稳定引起返工 0.82。（详见下图）

(为什么初期需求模糊反而是-0.45：等需求明确，写成文档，导致后期才能给明确需求，反而比早期给个模糊需求好，因更可能引起返工。)

	需求和设计在sprint初期很模糊，需要较长时间引导客户需求	需求和设计在sprint初期没给出，要比较后才能提供文档。	由于需求和设计不稳定导致的返工
sprint 30	1	1	1
sprint 31	2	0	1
sprint 32	4	1	1
sprint 33	2	2	0
sprint 34 和测试缺陷的相关性	4	1	1
	0.5079	0.6570	-0.4502
	0.8213		
Spikes中包含的需求量是否跟需求量一致			由于需求和设计不稳定导致的返工
sprint30	1	需求和设计在sprint初期很模糊，需要较长时间引导客户需求	需求和设计在sprint初期没给出，要比较后才能提供文档。
sprint31	2	0	1
sprint32	4	1	1
sprint33	2	2	0
sprint34	4	1	1
平均数(包含缺陷)	0.5079	0.6570	-0.4502
	0.8213		

若能写好需求，减少插入任务和变更，可以提升质量（降低测试缺陷数）。

改进措施

- 固化 Clarification 流程。当前是较为随机，由开发人员发起（统计数据中每个迭代 0~2 次），后面优化 Clarification 过程，要求每次迭代都必须做需求 Clarification，形式也更正式，明确甲乙双方哪些岗位、什么角色必须参与，在确认新过程有效后，就把过程固化下来。
  - 把需求 Clarification 要具备的重点写成检查单，增加检查项，如：
    - 需求是否完整明确
    - 是否按简化功能点方法写清行为、实体
    - 场景是否明确
    - 能否有对应明确测试用例（是否可测试）
  - 请甲方尽量减少插入任务和变更。

改进效果

除了客户满意度得到提升外，冲刺生产率也提升，例如：

生产率之前四分位数是(1.07, 1.13, 1.16), 采取控制插入任务, 减少变更, 做好 Clarification 等措施后, 有明显改善, 变为(1.04, 1.05, 1.08)。

(注：它们生产率 = 人天/功能点数，所以生产率系数是越低越好)

## Chapter 29

### C: 简化功能点 (SiFP) 简介与实例