我：你们管理层和客户都比较关心项目的进度，项目是否能按时完成？请问你们过去的项目如何？
开发：我们现在就是走敏捷开发，两周一个迭代。每次迭代前，我们聚一起开会，把所有用户故事按优先级排序，估
我：你们都能按计划在冲刺迭代里把所有任务开发好吗？
开发想了一下，说：我们也有延误出现，有些模块不能在计划的冲刺里完成。
我：为什么？
开发：我们花了很多时间修正系统测试暴露的问题，有一些是因为前面没有把需求分析透导致的返工。
我：请问你们是怎么估计每个任务应该花多少时间？
开发：我们会一起讨论，然后用敏捷的扑克牌方式，集思广益，估计出来。
我：除了你们每天站立会议监控项目的进展，你们要不要统计一下项目延期的水平是多少？
开发：其实我们也不算是延误，有些模块不能在本迭代完成，就放在下一个迭代去做。
我：软件开发有一个系数叫每人的生产率，就是开发人员一天能产出多少有效代码？你们有统计吗？
开发：你说的是否是敏捷开发燃烧图里的速度 (Velocity)？我们有画，但发现变化波动很大，所以后面我们也没有再
从以上对话可以看到，敏捷开发不一定能帮助团队按期交付，开发人员也不清楚自己的生产率是多少？

如何降低项目进度偏差

从以上对话可以看到，敏捷开发不一定能帮助团队按期交付，开发人员也不清楚自己的生产率是多少。要减少项目的延误，首先取决于估算是否准确。但如果只是依赖个人经验、头脑风暴去估计，很可能低估，因为可能没有考虑开发以外的一些因素，比如缺陷问题、需求问题等等。在同一个冲刺里，不能交付所有计划的模块，等同于延误。要做好估算，就需要有数据。数据需要从个人从收集的历史数据作为参考。IT 公司是否有注意这方面？我们可以从他们的新员工入职培训探索一下。

我：请问你们如何培训新入职的开发人员？
内部培训师：我们会先上一些基础的理论课，培训公司的代码规范、框架和复用的模块。然后进行个人或小组练习，
我：他们会收到什么反馈？
培训师: 理论培训后会有一些选择题考试，会出成绩。我们也会对他们做出来的程序打分。
我: 怎样打分？
培训师：我们依据评判标准打分，分数从 1-5,1 最低,5 最高。
我: 评判的标准可以说说吗?
培训师:(想了一下) 依据他们是否掌握了培训内容的重点打分。
我: 你们公司不是已经开始推行量化项目管理, 是否应该也配合量化管理, 不仅仅靠老师主观判断打分？
培训师：可否举些例子？
我: 例如他们质量方面的缺陷数，如果他们有做单元测试或者评审相关阶段的缺陷排除，项目管理相关所花费的工时

从量化培训开始

从以上对话可以看出很多公司都没有与量化软件开发项目管理相关的培训，这样如何能希望他们在项目中做好量化管理？入职不仅需要培训开发的技巧、怎么写好程序、做好面向对象设计，也应该学到要一直度量自己的开发过程，包括所花的工时，过程中发现的缺陷数等。
培训师：在我们新员工培训时，如何可以加入这些量化的元素？
我：很简单。首先，在先培训质量相关的技术指标，比如什么叫质量成本，什么

叫排除率等，然后做实战练习时，要求他们除了写程序，还需要记录一下所花的工时，自己项目走查时发现的缺陷数，测试的缺陷数，评审所花的时间等等。也要求开发人员除了提交程序以外，提交一个开发计划报告，内容包括实际所花工时，本来预估的工时，过程里发现得缺陷数，在哪一类过程等等。帮助他们一进入公司就培养一些量化度量的习惯，以避免过了试用期，开发的习惯已经养成，后面是很难改变的。

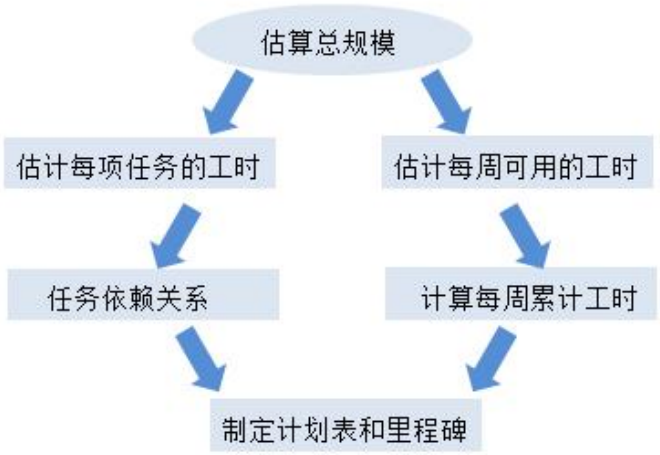
互动培训

举例：敏捷的小组传球游戏：
开头让人员传球，每做完一轮以后再估计下一轮的可以传多少个球。道理一样，让团队可以有据可依，更好地估计下一轮迭代。

本来这个传乒乓球的游戏，目的是让团队参考上一个迭代的数据，估计下一个迭代。但因为每个迭代开发的内容不一样，例如，有些模块是复用，有些是重新开发，肯定所花的工作量不一样，所以虽然这个用以往经验估算下一个的概念没有问题，但对实际上做好软件开发估算的作用很有限。如何可以更具体的帮团队做好估算监控？我们可以看看下面我的亲身实验。

从估算、策划到监控

挣值分析 (Earned Value EV) 是常用的项目监控技巧，没有项目管理工具也可以简单手动做用挣值分析。Humphrey 先生出版了不少书，他都用 EV 分析来监控进度，他在 PSP 书中以此为实例，介绍了如何用挣值分析策划与监控。



下面我也用同样思路来监控本书的进度：
{| class=”wikitable” |- | 今天 11 月 8 号，离年底要交书的全部初稿给出版社时间不到两个月，书的内容也完成了超过一半，但剩下的时间因为年底密密麻麻很多评估，而且评估就是早上九点开始到下午五六点结束，只有中间一些空档

时间和周末可以自由安排，所以我就用他的思路来策划估算是否能在年底完成要求。第一步算出现在到年底可以用在写书的时间，按每周估计:

	计划工时	累积工时
	Plan Hrs	Cum. Hrs
22.11.7	4	4
22.11.14	24	28
22.11.21	38	66
22.11.28	16	82
22.12.5	21	103
22.12.12	38	141
22.12.19	26	167
22.12.26	21	188
23.1.2	38	226
23.1.9	14	240

第二步从历史的数据估算剩下的章节，每个章节需要多少小时，写出计划的完成顺序，然后依据原计划可以使用的每周时间，按最佳使用算出每个章节可以在哪周完成，写在表格的右面空白处，从而就可以估算出完成所有章节何时可以完成，如果确实不能到 12 月底完成，就可能要重新调整、策划，是否有一些章节优先级比较低，暂时不放在第一版。

因觉得单点估算很难定一个点数，便用三点估算法。每一行估计都有三点：最差、最佳、最可能，用 PERT 方程式估计预计 (Expected) 和标准差 (Sigma)。:（实际顺序可能会有变，以上是已调整版本，原本 MGR 章节是放在后面。。。）

挣值析法所有东西 – EV, PV, AC 都是用钱来结算（这个例子我们就是用所花

工时），例如我们估计了每个章节所需要的工时，利用三点估算计算出每个章节预计的工时，加起来得出预计总工时是 80.33 工时（用 PERT 公式计算）。例如第一个章节 TDD，预期工时是 2.5，除以 80.33 就得出 0.031，我们就用 3.1 作为 TDD 的 PV（所有任务完成的总 PV 大概是 100）。同样方式算出 MGR 章节的 PV 是 6.6。（我本来也犯错，以为 PV 是和产出多少，或页数，成比例，但如用页数作为 PV 单位的话，就跟其它挣值分析法的系数，如 EV，对不上了，导致错误。）得出每一个章节对应的 PV 后，我们就可以计算累计的 PV 数:

Task 章节	最佳	最可能	最差	Expected	Sigma		PV	Cum. PV
TDD	2	2.5	3	2.50	0.17	0.03	3.1	3.1
MGR领导的任务	4	5	8	5.33	0.67	0.07	6.6	9.7
PSP	3	4	5	4.00	0.33	0.05	5	14.7
M6	4	5	6	5.00	0.33	0.06	6.2	20.9
M5	4	5	8	5.33	0.67	0.07	6.6	27.5
IC	4	5	6	5.00	0.33	0.06	6.2	33.7
CM	8	9	12	9.33	0.67	0.12	11.6	45.3
SiFP upd	3	4	5	4.00	0.33	0.05	5	50.3
INNOVATE创新	4	5	8	5.33	0.67	0.07	6.6	56.9
MitOCW写Python	2	2.5	3	2.50	0.17	0.03	3.1	60
克服拖延症	2	2.5	3	2.50	0.17	0.03	3.1	63.1
CI	6	8	12	8.33	1.00	0.10	10.4	73.5
RDM	6	8	12	8.33	1.00	0.10	10.4	83.9
Opening开章	3	3.5	4	3.50	0.17	0.04	4.4	88.3
最终检查	8	9	12	9.33	0.67	0.12	12.6	100.9
								100.9
		70.5	Sum_Exp	80.3333333		0.9024896		
						Overall Sigma		

估计哪一周可以完成哪些章节，得出总体的进度计划。例如第三周累积有 66 工时，应可完成总共 12(=1+4+7) 个章节，因用 PERT 三点估算，累积工时 95% 范围是（55.8 ~ 62.5）（详见附件）。

		95%区间（工时）		计划工时	累计工时		策划价值	累计策划价值	计划哪一周
	章节	下限	上限	Plan Hrs	Cum. Hrs		PV	Cum. PV	Plan Wk
22.11.7	TDD			4	4		3.1	3.1	1
22.11.14	MGR领导的任务			24	28		6.6	9.7	2
	PSP						5	14.7	2
	M6						6.2	20.9	2
	M5	20.0	24.3				6.6	27.5	2
22.11.21				38	66				
	IC						6.2	33.7	3
	CM						11.6	45.3	3
	SiFP						5	50.3	3
	INNOVATE创新						6.6	56.9	3
	MitOCW写Python						3.1	60	3
	克服拖延症	48.1	53.6				3.1	63.1	3
	CI	55.8	62.5				10.4	73.5	3

监控

按实际完成的情况就可以填右面 EV 挣值那栋，挣值的算法很简单，必须整个章节完成才算有挣值，不接受部分完成。比如第一周虽然 TDD 章节已完成九成，但实际是到第二周头才完成，所以第一周的挣值为 0。到了第二周才把本来的第一章节完成，所以里面才放上它的 PV(=3.1)。监控的好处，好比要准备半年后跑马拉松，开始长距离慢跑 LSD 锻炼，再进一步做连续马拉松配速训练，每周三天，每次 5 到 15 公里，速度是多少？比如平均每公里 6:25 分钟，有了数字才知道现状与标准的差距，才有动力对自己说现在离比赛不到 12 周，必须加强锻炼才能赶上。

要让项目有更大的机会按期完成，便需要监控，一直可以预计自己离最后交付差距多少？如果按现在的进展要到哪一周才可以完成，应都可以从数字预计出来。但是我们写书也靠灵感，跟写程序类似，可能顺序不一定按本来的执行，怎

么办？
其实道理也一样，你可以按新的顺序更新一下本来的计划，也把实际的填上。如果你是用简单的 Excel 表，这个改变可能只花几分钟，很快就能调整好顺序。（例如，本来现在看到的第二章节，本来计划放在后面才写，但因刚好预客户接触，有灵感，便放在前面写了。）

	章节	计划工时 Plan Hrs	累计工时 Cum. Hrs	策划价值 PV	累计策划价值 Cum. PV	计划哪一周 Plan Wk	实际工时 Actual Hr.	实际哪一周 Actual Wk.	挣值 EV	累计挣值 Cum. EV
22.11.7	TDD	4	4	3.1	3.1	1	2	2		
22.11.14	MGR领导的任务	24	28	6.6	9.7	2	7		3.1	3.1
	PSP			5	14.7	2	5.5			
	M6			6.2	20.9	2				
	M5			6.6	27.5	2				
22.11.21		38	66							
	IC			6.2	33.7	3				
	CM			11.6	45.3	3				
	SiFP			5	50.3	3				
	INNOVATE创新			6.6	56.9	3				
	MitOCW写Python			3.1	60	3				
	克服拖延症			3.1	63.1	3				
	CI			10.4	73.5	3				
22.11.28		16	82							

要估算任务工时便要依赖以往类似活动的历史记录，如果不是每天记录，过后无法记得住。每周依据个人的习惯，每天记录实际所花小时，然后统计每周累计。但很多人都没有这个习惯，如何养成这个习惯？（详见附件 Tips）
|}

总结

很多软件开发项目延误（或到交付前团队通宵达旦加班）的原因是因为以前没有收集历史数据，导致估计所需人时都很理想。我也有同样问题：从我的挣值分析实例看到，因我是第一次写书，之前也没有养成统计章节所花的实际工时的习惯，所以在 11 月初做估算时，理想地认为可以最多花一个月完成，但过了 3 周就发现，实际比本来计划延误很多，原因包括：

- 实际可用工时没有想象这么理想，有很多意外事情都要处理。
- 低估了完成一章节所需时间：
  - 例如第一个章节 TDD 实际所花的工时与估计差不多，原因是这个章节是现有的，只是做了一些调整，所花时间不多；
  - 但到第二个章节 MGR 就不一样了，全部内容都是重新写的，也是从实际的客户现场、场景总结出来，如何与理论结合也很花心思。

所以下次当你遇到一些团队项目延误，可以问组长，或者项目经理，如下问题：

1. 如何估算项目工期
2. 如何估算每一项任务所需的工时，依据什么？
3. 如何收集实际工作量
4. 如何监控累计到现在的完成情况
5. 有没有统计以往的历史数据？如何用来帮助做好估算？

他可能说都有做，当你深入探索时，会发现很多类似我前面挣值分析、监控和估算等问题。

应该如何改善

软件开发和工业生产不同，工作量数据必须靠个人自己收集，是一个习惯的问题。所以 Humphrey 先生在 90 年代，推出了 PSP 个体软件过程，教导开发人员如何一步步自己收集数据、自己估算、自己监控，从最基础的记录花多少工时、发现多少缺陷、完成多少代码开始，逐步改善、提升，最终达到管理质量成本，尽量减少缺陷返工。详见附件 PSP 简介。

也正因为大部分的团队都没有收集数据的良好习惯，没有的 PSP 的基础，所以就很难要求他们在冲刺回顾时拿出数据，做根因分析，做下一轮的改善，不能从一个定性的管理升为以数据说话、量化管理的状态。反之，当开发人员养成了这习惯，他们就会更清楚知道自己的速度水平和质量水平，不会出现开头说那种情况，对自己的开发质量水平或生产率一无所知，只说已把上级安排的任务尽力做完。
有些企业深知量化管理对企业发展很重要，所以在新员工培训时，不仅仅教他们技术技能，也教他开始自己每天记录工时、缺陷数等习惯。因为新员工入职后，工作习惯就开始养成，后面要改变他们习惯很难。他甚至会觉得既然周边的人都没有这么做，为什么他要统计数据。

Q&A

资深敏捷顾问：我大约接近 20 年前对这个东西比较熟，PSP 里边一个包含两个重点：

1. 关于个体的估算的内容，这个我看您的文章里边表达的比较充分，
2. 关于缺陷记录分类统计和根因分析的内容，这一块实际上相当于是 cmmi5 级里边需要个体配合的地方。我看您几乎没有提到。

我：非常赞同，若要做到本手册后面里提到基于缺陷数据的量化根因分析并改进就要依赖个人记录缺陷与返工工作量，碍于篇幅有限，这里先用人时的估算与监控带出 PSP 的思路。

顾问：我自己也是在某军工航空项目里边用过一次。我们当年计划少跟踪多，大家本来就缺少生产率的概念，所以其实也不知道每天能干什么事，但只要记下来每天干了什么事，然后再往后比照的这个基数做就可以了。无论突然更快了或者更慢了，都需要看看（回顾）到底是为什么。

我：您觉得这种方法有用吗？

顾问：因为开发是需要创造力的，写新程序其实很难准确估计所需时间，但我后来发现如果估算是由组长负责去做，就能起很大的作用。例如有一次，我看有一个程序员花了一个月时间用 java 写了某模块，接近一万行代码。虽然我不熟悉 Java 语言，但觉得代码有点问题。后面我就找另外一个比较资深的组长，三位一起看代码。评审并删除无用的代码，最后剩下不到一百行代码就可以实现了。如果当初我们先做好这模块的估计，就可以避免一个月人时的浪费。所以我建议还是需要估算，但是应该有有经验的技术组长负责。

我：很好，你很熟悉敏捷，例如 Scrum 里用扑克牌方法估算，你觉得怎么样？

顾问：我先跟你讲个故事：我参加某冲刺估算会议，它们用扑克牌方法都某功能估算：“数据显示 “，其中一位估计是 8，另外一位估计是 40。为什么会差这么多呢？第一位解释只需要展示数据, 确实该工作量不大。但第二位理解就不一样了，包括整个数据的分析、输入和展示等，整个过程工作量确实很大。从这里可以看出，如果没有统一理解需求，就难以估算。所以那次以后，所以需求分析应分成 “实体” 和 “行为”，清晰地写出功能需求，减少误会。

我：赞同。Humphrey 先生在 PSP 书里强调，要参考历史数据来做好估算，其实也可适用于敏捷估算。不应但靠人的主观判断。

顾问：是的，所以我前面强调必须要有经验的主管去负责估算。估算很重要，但是要程序员自己去估很难。所以 PSP 二十年前的思路，还能适用于现在的敏捷开发团队，帮助他们做好估算。

PSP 里强调要评审设计、代码并分析缺陷，当时因为只靠人手，所以会很耗时。现在，我们有类似 SONAR 的静态扫描工具，就可以像机器人一样，做本来耗时的代码评审工作。但 SONAR 只能针对一些基本语句问题，针对整个 OO 设计，我们会建议用我们的 CCI 工具去扫描（例如，查看有没有类过大等问题），补充 SONAR 扫描的不足。跟 PSP 代码评审的思路一样，所有扫描出的问题都必须修正。有些团队觉得问题太多了，只处理掉那些重大的问题，剩下一些可能不会直接影响到功能的问题暂时不处理。我不赞同这思路，这么多的问题其实都是累积出来的，如果从一开始都一直有定期处理清空，就不应该累积大量问题，而且这些问题遗漏下来还是对程序有隐患。

我：很赞同，归根到底还是开发人员缺乏保证质量的意识。最近有个团队也是用工具扫描代码，然后我问他为什么找出的部分问题不处理。他说例如定义某个变量，但是没有用，虽然被扫出是问题，觉得不影响程序的运行就不处理。我解释说，这样就好像放了一个地雷，后面还是会爆炸的，所以还是应该要处理。

PSP 要求评审代码，分析每个发现的缺陷，讨论原因，然后后面改进。所以有些人认为 PSP 成本太高，只适用于高价值高质量要求的项目里（或作为个人修炼，短暂的时期内使用）
有很多自动化开发工具，可以利用它们更好实现 PSP 的原理，尽量不花太多人的时间，同样可以达到提高软件质量目的。
估算很重要，要做好，不仅仅要参考历史数据, 也需要对需求有共同的理解。
功能点方法是估算功能需求规模的一种标准，下篇会用些实例介绍如何做功能点估算。

附件

挣值分析法（**Earned Value**）

挣值分析法（Earned Value），用最简单的例子，来说明挣值分析中 PV、EV、与 AC 的意义。

PV：计划完成多少
AC：完成工作的实际成本是多少？
EV：实际完成了多少工作?

公式：

进度偏差 SV = EV - PV , SPI = EV / PV 完成占计划完成的%
成本偏差 CV = EV - AC , CPI = EV / AC 完成的价值占实际已花成本的%

本文实例主要关心进度偏差，所以没有计算 AC

例如：截止到 22.11.20 ，PV 累加本应是 27.5，EV 只有 3.1

所以进度偏差 SV = EV - PV = 3.1 - 27.5 = -24.4 工时

用 PERT 公式估算头三周任务累积工时的 95% 区间（工时）：
不用 MonteCarlo 模拟，直接把 12(=1+4+7) 章节任务的 ExpectedValue 加起来
计算 12 步总方差：（方差 = $Sigma^2$）
: 总方差 = 每步方差的总和
总 Sigma(=1.69) σ= 总方差的平方 (Sq.Root)

EXCEL 公式为：SQRT(SUM(所有方差范围)) = SQRT(SUM(H3:H15))

95% 范围下限 (=55.8)
EXCEL 公式为:SUM(所有 Expected 范围)-2* 总 Sigma = SUM(F3:F15)-2*I15
95% 范围上限 (=62.5)

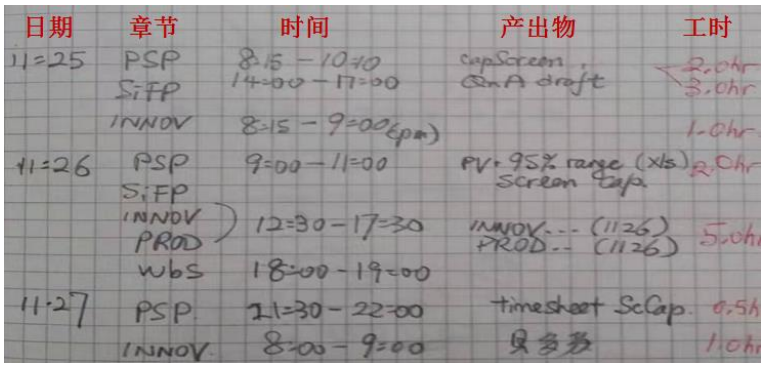
EXCEL 公式为: SUM(所有 Expected 范围)+2* 总 Sigma = SUM(F3:F15)+2*I15

								95%区间（工时）	
章节	最佳	最可能	最差	Expected	Sigma	Sigma^2		下限	上限
TDD	2	2.5	3	2.50	0.17	0.027778	0		
MGR领导的任务	4	5	8	5.33	0.67	0.444444	0		
PSP	3	4	5	4.00	0.33	0.111111			
M6	4	5	6	5.00	0.33	0.111111	0		
M5	4	5	8	5.33	0.67	0.444444	1.07	20.0	24.3
IC	4	5	6	5.00	0.33	0.111111			
CM	8	9	12	9.33	0.67	0.444444			
SiFP	3	4	5	4.00	0.33	0.111111			
INNOVATE创新	4	5	8	5.33	0.67				
MitOCW写Python	2	2.5	3	2.50	0.17	0.027778			
克服拖延症	2	2.5	3	2.50	0.17	0.027778			
CI	6	8	12	8.33	1.00	1	1.69	55.8	62.5

Tips：如何简单记录工时

如前面“克服拖延症”里提到，每人根据自己的习惯，采用每日 ToDoList 的方式，然后在开始前，简单记一下时间，结束时也记下时间。然后每日/每周统计每任务总工时:

550px



（如只是个人，不一定需要项目管理工具；如果是团队，可把自己手工记录上传到项目管理工具，方便团队记录与监控。）

PSP - 个体软件过程 Personal Software Process 简介

PSP 的简单介绍

- **PSP0** 基础 - 工时：计划与实际对比；每阶段引入多少缺陷；排除了多少缺陷
- **PSP0.1** 加入代码行统计 - 计划与实际对比；代码规范
- **PSP1** 加入使用 PROBE 方法做规模估算
- **PSP2** 加入设计与代码评审的计划与统计
- **PSP3** Cyclic process - 先做策划与总体设计，然后多轮开发；有点类似迭代开发。

PSP 跟 CMMI 成熟度模型类似，也是按部就班一步步，帮助软件工程师利用度量改进自我的开发过程。

PSP 0.1

第一步 PSP 先估计写某个模块所需要的时间和实际花的时间，记录所有的缺陷，包括因为需求、设计或者实现引起的问题，记录修复时间，从开始发现问题直到缺陷被解决，提升到 PSP0.1 策划不仅仅是估计所需时间，加入了规模的概念，本来 PSP 是用代码行数来估计规模，也可以使用简化功能点方式估计规模大小。
在 PSP0.1 的阶段还没有用规模做策划、估算，只是记录实际的规模，里面包括基本规模，就是在开发之前软件系统本来有多大。也记录删除、改变、增加、复用的规模数等等。最后总的规模数应该是等于基本、新增、删除、复用相加。

PSP1

下一步，PSP1 就有规模计划的概念，跟依据 0.1 的规模定义一样，我们在做开发之前，先估计刚才的所有规模数，并用回归方程估算对应的工作量或者进度，包括偏差的范围。
在 PSP0.1 增加了一个过程改进建议的环节，依据上一次迭代的数据，发掘下一轮可以完善的改进过程。也加入了编码标准，这个跟我们前面章节提到代码规范的概念相同。所以在 PSP0.1，除了计划，也需要有一个叫过程改进经验教训的部分，每一轮都应该有复盘回顾的概念，来提升个人个人的开发过程。

PSP1.1

PSP1.1 基于的 PSP1 的基础，加入挣值分析法去监控实际的项目进展，加入了 PV 和累加的 PV，计划多少和 EV 挣值和累计的挣值，反应实际完成多少。从那些系数也可以算出一个叫 CPI 成本性能指数，来反应完成与计划怎么比较，是否有延误，预计什么时候可以完成。详细可以看用挣值分析法估计写书完成时间的实例。

PSP2

到 PSP2 就开始增加评审的概念，包括设计评审、代码评审。这些同行评审是很有效的方式，避免缺陷等到出事才暴露、出现问题，因为如果只依赖测试暴

露缺陷的话，只是告诉你这个程序跑不通，你还是要看代码才知道问题在哪里，怎么修正？但评审就直接看代码，首先它可以让很多缺陷在评审就发现，就不要等到测试才暴露问题。评审也可以找出一些不一定测得出来的问题，例如有些银行的系统质量要求很高，核心的算法不能有任何错误，他们都会要求核心模块必须走专家评审，因为他们也知道很多一些核心算法的问题不能单靠测试来发现和处理。也是这个原因，很多公司也会要求代码必须走静态扫描，利用扫描机器人评审代码，预先发现一些明显代码违规的语句问题，弥补单靠人手去评审衡的不足。
有了 2 的基础，2 也开始加入缺陷密度的概念，即缺陷数除以模块的规模大小，本来 PSP 是使用代码行的，我们也可以用功能点取代代码行，因为单看缺陷数无法比较，无论个人自己或者整个团队或者团队之间，所以必须除以规模大小，才可以把系数归一，变成一个项目组之间，人之间可以比较的系数。也引入了排除率的概念，如果我们只是做了评审，但是都是 0 发现，你的缺陷排除率就是零了，一点效果都没有，所以排除率可以看成是当前评审发现的缺陷数除以当前系统内总共引入的缺陷数，比率越高越好。也有阶段或者过程的排除率，就是这个阶段开始时总共有多少缺陷，然后在这个阶段里我们找出多少，作为分子的一个比例。还有一个就是缺陷排除的速度，按每小时评审的时间找出多少缺陷来判断评审中，可以找出缺陷的速度有多快，评审的效率有多高。

评审质量是 PSP2.1 的概念，2.1 也增加质量成本（COQ）概念，与前面敏捷回顾篇里强调软件开发缺陷越后发现，返工量会几何式上升的思路一致。

PSP3

接近敏捷迭代的思路，把程序分成不超于几百行的模块，然后把模块按优先级分到不同的迭代，但还需要有需求与设计。敏捷增加了精益的思路，因为需求可能会有变，所以可以暂时不做后面迭代的估算，只先针对当前的迭代去做策划、估算。敏捷里面的迭代回顾跟 PSP 的回顾如出一辙。
大家可以参考 PSP 书里面的教材和练习，比如在书中要求学员自己编写程序，自己记录时间。虽然现在开发语言统计方法跟 90 年代有差异，但原理还是共通的。如果对新员工难以安排学校式的一周课程，也可以考虑用自学的形式，按要求去记录学习、写报告。

学 PSP，培养好习惯所以编程人员可以用 PSP 的原则，要他们记录。因为现在是写程序，不是玩游戏，所以更贴近实际。也要求后面的程序，让他分析自己的返工、质量成本、缺陷排除等，让后面项目团队回顾时有真实数据做分析。
例如在新员工培训时，在练习中，要求他们有编码规范的概念，按照公司的规范参考来写程序，在每一次做开发之前，都先简单计划一下时间和预估缺陷数。开发完后，记录实际的时间与缺陷数，因为虽然不仅仅是写一个程序，是一个系列，在做下一次开发时，可以参考以往练习的数据来更好估计，会看到自己的估计越来越准。

References

1. Humphrey, Watts S.: "A Discipline for Software Engineering"
2. Humphrey, Watts S.: "PSP: A Self-Improvement Process for Software Engineers" (2005)
:::---===<<< END >>>===---

我：你们高层主要关注设计和编码人员哪些点？怎么才算是做好开发的工作开发的工作？
开发：有没有遵守公司的编码规范？有没有客户投诉？高层就是关注这些了。
我：有没有一些数据可以反映你们的表现？让我可以在高层面前表扬你？
开发：比如我们开发都能按计划完成，没有延期。
我：除了按要求时间交付以外，你们公司不是也很注重产品质量的吗？有什么数据可以反映你们的开发质
开发：我们代码走查发现的缺陷都在项目经理的计划范围之内。（他们公司开始推行量化管理，利用模型
我：缺陷数在范围之内就算是质量好了吗？如果我们要提升产品的质量，是否应该找查评审的缺陷数少一
开发想了一下：应该是少一点好，但也不能太少。
另外一位开发：应该是找出的缺陷多一点才算好。
我：从这个问题会不会发现，我们难以单靠代码走查的缺陷数来判断产品的质量？整个软件开发产品质量
我：所以我们不能单纯从走查发现多少缺陷来衡量质量，更应该看缺陷排除率。通过走查缺陷占当前缺陷

如何衡量开发质量

从以上跟开发的对话，可以了解很多开发人员人都没质量概念，不理解如何通过量化衡量开发的质量，量化管理开发的质量。所以我们除了培训开发人员写程序，也要教怎么做好度量，收集开发相关的工作量缺陷数，让开发团队知道自己的开发质量，有什么不足，需要后面完善。这些需要先从个人的习惯开始，但习惯不会自然养成，也不会单靠听一些理论课形成，必须在培训时，加上一些自己动手的编程练习，学习怎么记录自己的开发相关数据，并分析了解自己开发的生产率、速度、质量。

How PSP agile XP version

detail 'PSP' like training program

028

从以上对话可以看出很多公司都没有与量化软件开发项目管理相关的培训，这样如何能希望他们在项目中做好量化管理？入职不仅需要培训开发的技巧、怎么写好程序、做好面向对象设计，也应该学到要一直度量自己的开发过程，包括所花的工时，过程中发现的缺陷数等。
培训师：在我们新员工培训时，如何可以加入这些量化的元素？
我：很简单。首先，在先培训质量相关的技术指标，比如什么叫质量成本，什么叫排除率等，然后做实战练习时，要求他们除了写程序，还需要记录一下所花的工时，自己项目走查时发现的缺陷数，测试的缺陷数，评审所花的时间等等。也要求开发人员除了提交程序以外，提交一个开发计划报告，内容包括实际所花工时，本来预估的工时，过程里发现得缺陷数，在哪一类过程等等。帮助他们一进入公司就培养一些量化度量的习惯，以避免过了试用期，开发的习惯已

经养成，后面是很难改变的。

029

我：下面给你介绍一下，通常我们帮企业新员工培训的一些重点？例如在练习时，要求他们有编码规范的概念，按照公司的规范参考来写程序，在每一次做开发之前，都先简单计划一下时间和预估缺陷数。开发完后，记录实际的时间与缺陷数，因为虽然不仅仅是写一个程序，是一个系列，在做下一次开发时，可以参考以往练习的数据来更好估计，他会看到自己的估计越来越准。敏捷有一个小组?? 游戏，开头让人员传乒乓球，每做完一轮以后再估计下一轮的可以传多少个球。道理一样，让团队可以有据可依，更好地估计下一轮迭代。所以编程时可以用同样的原则，要他们记录。因为现在是写程序，不是玩游戏，所以更贴近实际。也要求后面的程序，让他分析自己的返工、质量成本、缺陷排除等，让他们后面参与项目团队的回顾，分析数据的时候，已经掌握了这些概念，并能在回顾时积极参与讨论。

030

我：你们高层主要关注设计和编码人员哪些点？怎么才算是做好开发的工作开发的工作？
开发：我们有没有遵守公司的编码规范？有没有客户投诉？高层就是关注这些了。
我：有没有一些数据可以反映你们的表现？让我可以在高层面前表扬你？
开发：比如我们开发都能按计划完成，没有延期。
我：除了按要求时间交付以外，你们公司不是也很注重产品质量的吗？有什么数据可以反映你们的开发质量？
开发：我们代码走查发现的缺陷都在项目经理的计划范围之内。（他们公司开始推行量化管理，利用模型预计评审的缺陷数范围。）
我：缺陷数在范围之内就算是质量好了吗？如果我们要提升产品的质量，是否应该找查评审的缺陷数少一点？
开发想了一下：应该是少一点好，但也不能太少。
另外一位开发：应该是缺陷多一点质量好。
我：从这个问题你们会不会发现，我们难以单靠找查的缺陷数来判断产品的质量？整个软件开发产品质量，如果最终都没有任何 BUG 被客户发现，就表示这个产品质量好了吧？开发：应该是的。
我：所以我们不能单纯从找查发现多少缺陷来衡量质量？更该看缺陷排除率。通过走查在当前缺陷的百分比？这个百分比越大，表示找查的效果越好。但这个只是其中一个过程里面的参考数，软件开发有个特点，缺陷越是到了测试阶段暴露，返工的成本越高，但是发现得越早，比如评审时，成本或工作量都少，可能半个小时就能搞定。而到了验收才暴露的话，可能需要几个人天。时间差异很大所以。除了缺陷排除率，也需要看质量成本，总共花在返工、修改 bug 的成本是多少？越多就表示越需要改进，但要注意这些系数都是过程中间的一些质量参数，可以帮团队了解中间过程做得如何。但最终看遗漏到最终客户的缺陷有多少，才是真正的产品质量。

031

从以上跟开发的对话，可以了解很多开发人员人都没质量概念，不理解如何通过量化衡量开发的质量，量化管理开发的质量。所以我们除了培训开发人员写程序，也要教怎么做好度量，收集开发相关的工作量缺陷数，让开发团队知道自己的开发质量，有什么不足，需要后面完善。这些需要先从个人的习惯开始，但习惯不会自然养成，也不会单靠听一些理论课形成，必须在培训时，加上一些自己动手的编程练习，学习怎么记录自己的开发相关数据，并分析了解自己开发的生产率、速度、质量。

0557

项目估算、策划到监控，如果没有项目管理工具也可以简单用挣值分析手动做。?? 先生在 PSP 书中以此为实例，介绍了如何用挣值分析策划与监控。下面我也用同样思路来策划与监控：
今天 11 月 8 号，离年底要交书的全部初稿给出版社时间不到两个月，书的内容也完成了超过一半，但剩下的时间因为年底密密麻麻很多评估，而且评估就是早上九点开始到下午五六点结束，只有中间一些空档时间和周末可以自由安排，所以我就用他的思路来策划估算是否能在年底完成要求。第一步算出现在到年底可以用在写书的时间，按每周估计；第二步从历史的数据估算剩下的章节，每个章节需要多少小时，写出计划的完成顺序，然后依据原计划可以使用的每周时间，按最佳使用算出每个章节可以在哪周完成，写在表格的右面空白处，从而就可以估算出完成所有章节何时可以完成，如果确实不能到 12 月底完成，就可能要重新调整、策划，是否有一些章节优先级比较低，暂时不放在第一版。

0558

在策划的时候也填上 PV 策划价值，例如比如第一个章节估计需要 15.1 小时，就初步定它的 PV 等同于 0.99，第二部分章节估计需要花 8.14 个小时，按比例它的 PV 等于 0.53，累计 1.52，然后累积到第二章节人时是 23.24。所以按预留的第二周，累积可以花 40 个小时，第一周累计只有 22 小时，所以估计第二个章节最快在周二才可以交付。类似地把每一行都填上，估计完成所有需要多少周，是否能在 12 月底前完成。
监控：每周依据个人的习惯，当成每天会有记录，实际所花的小时，例如前面写程序那样，大概记一下每天花了多少小时，也统计每周累计多少小时？按实际完成的情况就可以填右面?? 挣值的那栋，挣值的算法很简单，不接受一般完成或者九成完成，必须整个章节完成才算有相对的挣值，比如第一周第一个章节未能完成，第一周的挣值为 0。到了第二周才把本来的第一章节完成，所以里面才放上它本来的 PV0.99。这个监控有个好处，好比跑马拉松，现在跑了 24 公里的半马，你的速度比计划的延误了多少？有了数字你才知道自己的差距，才有动力对自己说现在已经滞后了，必须赶上。让整个项目有更大的机会按期完成，然后在整个进展过程中，一直可以预计自己离最后交付差距多少？如果按现在的进展要到哪一周才可以完成？都可以从数字预计出来。但是我们写书也靠灵感，跟写程序类似，可能顺序不一定按本来的执行，怎么办？
其实道理也一样，你可以按新的顺序更新一下本来的计划，也把实际的填上。如果你是用简单的 Excel 表，这个改变可能只花几分钟，很快就完成。如果你觉

得单点估算不够准确，希望用三点估算法也一样。每一个估计就有一个三点的范围，最大、最小、最高、最差的机会，你要估计总的，但这种就需要用一些??工具帮你实现估计。
我建议开始的时候而是先从单点开始，形成习惯后如果希望更准确、有范围，才开始使用三点估算法。

0559

PSP 个体软件过程跟 CMMI 成熟度模型类似，也是按部就班一步步，帮助软件工程师利用度量改进自我的开发过程。第一步 PSP 先估计写某个模块所需要的时间和实际花的时间，记录所有的缺陷，包括因为需求、设计或者实现引起的问题，记录修复时间，从开始发现问题直到缺陷被解决，提升到 SP0.1 策划不仅仅是估计所需时间，加入了规模的概念，本来 PSP 是用代码行数来估计规模，也可以使用简化功能点方式估计规模大小。
在 PSP0.1 的阶段还没有用规模做策划、估算，只是记录实际的规模，里面包括基本规模，就是在开发之前软件系统本来有多大。也记录删除、改变、增加、复用的规模数等等。最后总的规模数应该是等于基本、新增、删除、复用相加。
下一步，PSP1 就有规模计划的概念，跟依据 0.1 的规模定义一样，我们在做开发之前，先估计刚才的所有规模数，并用回归方程估算对应的工作量或者进度，包括偏差的范围。
在 PSP0.1 增加了一个过程改进建议的环节，依据上一次迭代的数据，发掘下一轮可以完善的改进过程。也加入了编码标准，这个跟我们前面章节提到代码规范的概念相同。所以在 PSP0.1，除了计划，也需要有一个叫过程改进经验教训的部分，每一轮都应该有复盘回顾的概念，来提升个人个人的开发过程。

0560

PSP1.1 基于的 PSP1 的基础，加入挣值分析法去监控实际的项目进展，加入了 PV 和累加的 PV，计划多少和 EV 挣值和累计的挣值，反应实际完成多少。从那些系数也可以算出一个叫 CPI 成本性能指数，来反应完成与计划怎么比较，是否有延误，预计什么时候可以完成。详细可以看用挣值分析法估计写书完成时间的实例。再到 PSP2 就开始增加评审的概念，包括设计评审、代码评审。这些同行评审是很有效的方式，避免缺陷等到出事才暴露、出现问题，因为如果只依赖测试暴露缺陷的话，只是告诉你这个程序跑不通，你还是要看代码才知道问题在哪里，怎么修正？但评审就直接看代码，首先它可以让很多缺陷在评审就发现，就不要等到测试才暴露问题。评审也可以找出一些不一定测得出来的问题，例如有些银行的系统质量要求很高，核心的算法不能有任何错误，他们都会要求核心模块必须走专家评审，因为他们也知道很多一些核心算法的问题不能单靠测试来发现和处理。也是这个原因，很多公司也会要求代码必须走静态扫描，利用扫描机器人评审代码，预先发现一些明显代码违规的语句问题，弥补单靠人手去评审衡的不足。
有了 2 的基础，2 也开始加入缺陷密度的概念，即缺陷数除以模块的规模大小，本来 PSP 是使用代码行的，我们也可以用功能点取代代码行，因为单看缺陷数无法比较，无论个人自己或者整个团队或者团队之间，所以必须除以规模大小，才可以把系数归一，变成一个项目组之间，人之间可以比较的系数。也引入了排除率的概念，如果我们只是做了评审，但是都是 0 发现，你的缺陷排

除率就是零了，一点效果都没有，所以排除率可以看成是当前评审发现的缺陷数除以当前系统内总共引入的缺陷数，比率越高越好。也有阶段或者过程的排除率，就是这个阶段开始时总共有多少缺陷，然后在这个阶段里我们找出多少，作为分子的一个比例。还有一个就是缺陷排除的速度，按每小时评审的时间找出多少缺陷来判断评审中，可以找出缺陷的速度有多快，评审的效率有多高。

0565

要估算任务，手画的工时便要依赖以往类似活动的历史记录，如果不是每天记录，过后无法记得住。但很多人都没有这个习惯，我如何养成这个习惯呢？
首先，每天记下今天主要任务，然后在开始前，简单记一下时间，结束时也记下时间。这个时间一般是从做培训得来的，因为做培训也要同样记录每次时间上要花多少在哪个模块中，然后可以后面微调，下一次同样培训的时间安排是否合适？当我做了 CMMI 培训十多次以后，基本上就不用每次记录了，基本每次所花的都比较稳定。刚开始的时候，肯定需要。每次活动所花的时间不太一样，比如我写文章，有些是复用的，新增的字不多就比较快，但是有些是重新写的，可能不仅是写，还要查相关的数据资料，这个就会比较慢。所以我们要用那些数据做后面的分析的话，可能也要区分复用多少，新增多少等等。自己一直做任务，还是应该比较有概念，但是如果给别人看，就不一定看得懂了。所以，还是要分开。这个道理也可以用于写程序的统计记录。

图

566

很多软件开发项目延误，或者到期交付前团队通宵达旦加班的原因，是因为以前没有收集历史数据，导致估计的时间、人时都很理想。无论你有没有做过软件开发项目，都可以从我如何使用挣值分析法监控写书进展。我是第一次写书，之前也没有养成统计章节所花的实际工时的习惯，所以在 11 月初做估算的时候，理想地认为可以最多花一个月完成，但过了 3 周就发现，实际比本来计划延误很多。原因包括实际可用工时没有想象这么理想，有很多意外事情都要处理，另外也低估了一个章节所需要的时间，第一个章节 TDD 实际所花的工时与估计差不多，原因是这个章节是现有的，只是做了一些调整，所花时间不多，但到第二个章节就不一样了，全部内容都是重新写的，也是从实际的客户现场、场景总结出来，如何进行与理论结合的编辑也很花心思。所以下次当你遇到一些团队项目延误，你可以问组长或者项目经理如下问题：1、如何估计项目的进度？2、有没有收集开发时的实际工时？

567

一、如何估算项目工期
二、如何估算每一项任务所需的工时，依据是什么
三、如何收集实际工作量
四、如何监控累计到现在的完成情况
五、有没有统计以往的历史数据，并用来帮助做估算

他可能初步回应你都有做，当你深入探索时，会发现很多类似我前面挣值分析、监控和估算的问题，或者甚至比我做的更差。
应该如何改善？软件开发和工业生产不一样，工作量数据必须靠个人自己收集，是一个习惯的问题。所以?? 先生在 90 年代，就推出了 PSP 个体软件过程，教开发人员如何一步步自己收集数据、自己估算、自己监控，从最基础的记录花多少工时、发现多少缺陷、完成多少代码开始，逐步改善、提升，最终达到有质量成本，尽量减少缺陷返工。详见附件简介。也正因为大部分的团队都没有收集数据的良好习惯，没有的 PSP 的基础，所以就很难要求他们在冲刺回顾时拿出数据，做根因分析，做下一轮的改善，不能从一个定性的管理升为以数据说话、量化管理的状态。当开发人员养成了这个好习惯，他们就会更清楚知道自己的速度水平和质量水平，不会出现开头说那种情况，对自己的开发质量水平或生产率一无所知，只求把上级安排的任务做完。
有些企业也深知量化管理对企业发展的重要，所以在新员工培训时，也不仅仅教他们一些技术的技巧。让他开始自己每天记录工时、缺陷数等习惯。因为如果入职时不能改变。

568

因为一般新员工入职后，工作习惯就已经养成，后面要他改变习惯，每天统计自己的数据是很难的，他也会觉得周边的人都没有这么做，为什么我要做统计，感觉很奇怪。

569

我：你们管理层和客户都比较关心项目的进度，项目是否能按时完成？请问你们过去的项目如何？
开发：我们现在就是走敏捷开发，两周一个迭代。每次迭代前，我们聚一起开会，把所有的用户故事按优先级排序，估计这个迭代可以完成哪些任务？然后放在大白板的待办事项里。每天我们会做站立会议，监控实际的进展。
我：你们都能按计划在冲刺迭代里把所有任务开发好吗？
开发想了一下，说：我们也有延误出现，有些模块不能在计划的冲刺里完成。
我：为什么？
开发：我们花了很多时间修正系统测试暴露的问题，有一些是因为前面没有把需求分析透导致的返工。
我：请问你们是怎么估计每个任务应该花多少时间？
开发：我们会一起讨论，然后用敏捷的扑克牌方式，集思广益，估计出来。
我：除了你们每天站立会议监控项目的进展，你们要不要统计一下项目延期的水平是多少？
开发：其实我们也不算是延误，有些模块不能在本迭代完成，就放在下一个迭代去做。
我：软件开发有一个系数叫每人的生产率，就是一个开发人员一天或者一周能产出多少？你们有统计吗？
开发：你说的是否是敏捷开发燃烧图里的速度？我们有画这个图，但发现它的变化或者波动很大，所以后面我们也没有再用。
从以上对话可以看到，敏捷开发不一定能帮助团队按期交付，开发人员也不清楚自己的生产率是多少？

570

要减少项目的延误，首先取决于估算是否准确？但如果只是依赖个人经验、头脑风暴去估计，很可能低估，因为可能没有考虑开发以外的一些因素，比如缺陷问题、需求问题等等。在同一个冲刺里，不能交付所有计划的模块，等同于延误。要做好估算，就需要有数据。数据需要从个人从收集的历史数据作为参考。IT 公司是否有注意这方面？我们可以从他们的新员工入职培训探索一下。

571

本来这个传乒乓球的游戏，目的是让团队参考上一个迭代的数据，估计下一个迭代。但因为每个迭代开发的东西不一样，有些模块是复用的，有些是重新开发的，肯定所花的工作量不一样，所以虽然这个用以往经验估算下一个的概念没有问题，但对实际上做好软件开发估算的作用很有限。如何可以更具体的帮团队做好估算监控？我们可以看看我的一个亲身实验。

572

也可以参考 PSP 书里面的教材和练习，比如在书中要求学员自己编写程序，自己记录时间。虽然现在开发语言统计方法跟 90 年代有差异，但原理还是共通的。如果对新员工难以安排学校式的一周课程，也可以考虑用自学的形式，按要求去记录学习、写报告。

replace by QnA on productivity 我：你们高层主要关注设计和编码人员哪些点？怎么才算是做好开发的
开发：有没有遵守公司的编码规范？有没有客户投诉？高层就是关注这些了。
我：有没有一些数据可以反映你们的表现？让我可以在高层面前表扬你？
开发：比如我们开发都能按计划完成，没有延期。
我：除了按要求时间交付以外，你们公司不是也很注重产品质量的吗？有什么数据可以反映你们的开发
开发：我们代码走查发现的缺陷都在项目经理的计划范围之内。（他们公司开始推行量化管理，利用模型
我：缺陷数在范围之内就算是质量好了吗？如果我们要提升产品的质量，是否应该找查评审的缺陷数少
开发想了一下：应该是少一点好，但也不能太少。
另外一位开发：应该是找出的缺陷多一点才算好。
我：从这个问题会不会发现，我们难以单靠代码走查的缺陷数来判断产品的质量？整个软件开发产品质量
我：所以我们不能单纯从走查发现多少缺陷来衡量质量，更应该看缺陷排除率。通过走查缺陷占当前缺

如何降低项目进度偏差

从以上跟开发的对话，voice 可以了解很多开发人员人都没质量概念，不理解如何通过量化衡量开发的质量，量化管理开发的质量。所以我们除了培训开发人员写程序，也要教怎么做好度量，收集开发相关的工作量缺陷数，让开发团队知道自己的开发质量，有什么不足，需要后面完善。这些需要先从个人的习惯开始，但习惯不会自然养成，也不会单靠听一些理论课形成，必须在培训时，加上一些自己动手的编程练习，学习怎么记录自己的开发相关数据，并分析了解

自己开发的生产率、速度、质量。

但实际上 IT 公司有多注重度量的培训？

573

挣值析法所有东西都是用钱来结算，在这个例子我们就是用所花工时，例如我们估计了每个章节所需要的工时，利用三点估算计算出每个章节预计的工时，加起来得出预计总工时是 80.33。例如第一个章节 TDD，预期工时是 2.5，除以 81.33 就得出 0.031，我们就用 3.1 作为 TDD 的 PV。同样方式算出 MGR 章节的 PV 是 6.6。（不要以为 PV 是和产出多少或页数成比例，因为用了页数作为单位的话，就跟其它挣值分析法的系数范围对不上了，导致错误。）得出每一个章节对应的 PV 后，我们就可以计算累计的 PV 数，估计哪一周可以完成哪些章节？得出总体的进度计划。例如第三周累积有 66 个工时，就可以在这周完成六个章节。

575

到了 PSP3 就接近敏捷叠代的思路了，但还是需要有需求与设计，把程序分成不超于几百行的模块，然后把模块按优先级分到不同的迭代。即使敏捷增加了精益的概念，如果有些是后面迭代的估算，因为可能会有变，所以可以暂时不估不做，只先针对当前的迭代去做策划、估算。
敏捷里面的迭代回顾跟 PSP 的回顾如出一辙。

576

很赞同你说的第二点，PSP 是非常好一个实战的方式，让开发人员自己记录数据，并开始理解自己的生产力水平和质量水平，不一定要增加额外的工作量，但是要改变习惯，就是记录，例如在开始做任何一个开发之前，先在本子上写一下开始时间，然后做完在本子上写上产出物和结束时间。然后在每周监控时跟计划对比，以我写书的经历，没有增多多少额外的工作量，只是习惯的改变，也很多人都没有记录实际时间和策划，然后对比的习惯，也是真正因为没有这个习惯，一直都不知道现在延迟多少。也因为不知道自己延迟，所以没有这个危机感或者压力也要加快速度，容易躺平。我估计你说成本高，可能是因为航天项目要求很多监控的报告报上去，就耗费你写报告的时间。但是假如我们不需要那些人花时间去写那些监控报告，而是只需要他们在迭代回顾时把那些统计的数据拿出来分析，就会大大的减少那些用于报告的时间，但是可以获得重要的生产力数据，可以在回顾或者根因分析时有数据支撑，可以作为未来数据的参考。个人有了这个习惯，也可以知道自己的编码的速度、质量的水平了。

600px

文件:psp1.jpg

2022.11.25

估算很重要，要做好，不仅仅要参考历史数据, 也需要对需求有共同的理解。功能点方法是估算功能需求规模的一种标准，下篇会用些实例介绍如何做功能点

估算。

577

我：您觉得这种方法有用吗？
顾问：因为开发是需要创造力的，写一个新程序其实很难准确估计所需时间，但我后来发现如果估算是由组长负责去做，就能起很大的作用。例如有一次，我看有一个程序员花了一个月时间用 java 语言写了一个模块，接近一万行代码。虽然我不熟悉 Java 语言，但觉得这个代码有点问题。后面我就找另外一个比较资深的组长，三位一起看他的代码。评审后删除，最后剩下不到一百行代码就可以实现了。如果当初我们先做好这个模块的估计，就可以避免一个月人时的浪费。所以我建议还是需要估算，但是应该有有经验的技术组长负责。
我：很好啊。你也很熟悉敏捷，例如 Scrum 里的估算方法，比如扑克牌估算，你觉得怎么样？
顾问：我先跟你讲个故事。有一次我参加一个估算的会议，有一个功能的数据，其中一位估计是 6，另外一位估计是 50。为什么会这么多呢？第一位解释只需要展示数据, 确实该工作量不大。但第二位理解就不一样了，包括整个数据的分析、输入和展示等，整个过程工作量比较大。从这里可以看出，如果没有同步理解需求，就会难以估算？所以那次以后，我就提倡需求应该分成实体和行为，清晰地写出来，减少这种误会。
我：赞同。先生在 PSP 书里面强调，要利用、参考历史数据来做好估算，其实也可适用于敏捷的估算的。不应当靠当时所有人的主观判断。
顾问：是的，所以我前面强调必须要有经验的主管去负责估算。估算很重要，但是要程序员自己去估很难。所以 PSP 二十年前的思路，可以适用于现在的敏捷开发团队，帮助他们更好的做估算。PSP 里面强调要评审代码并分析缺陷，当时因为没有太多供给的支撑，只靠人手，所以会很耗时间。到了现在，我们有类似 SONAL 的静态扫描的工具，就可以像机器人一样，代替本来耗时的代码评审工作。但 SONAL 只能针对一些基本语句问题，针对整个 OO 设计，我们会建议用我们的 CCI 工具去扫描，看有没有类过大等问题，补充 SONAL 扫描的不足。但总而言之，跟 PSP 代码评审的思路一样，少数的问题都必须修改好。有些团队觉得问题太多了，只处理掉一些重大的问题，剩下一些可能不会直接影响到程序的问题先不处理。我是不赞同的，因为那些问题遗漏下来还是对程序有隐患。你可以想想，这么多的缺陷问题都是累积出来的。如果本身有定期处理清空，就不应该累积大量问题。
我：很赞同。最近有个团队也是用算法扫描，然后我问他为什么找出的问题不处理。他说例如定义某个变量，但是没有用，觉得不影响程序的运行就不处理。我解释说，这样就好像放了一个地雷，后面还是会爆炸的，所以还是应该要处理。所以有了现在很多自动化开发工具，我们可以利用它们更好实现 PSP 的原理，可以尽量不花太多的人工时间，同样可以达到提高提高软件质量的目的。
我：你们团队（或个体）是否逐渐便有了自身生产率标杆的概念，开始了解自己的水平？
顾问：是的，但 PSP 的实施成本还是非常高的。有两种场景可以使用:

1. 场景 - 像刚才说的，在高价值高质量要求的项目里边
2. 场景 - 作为个人修炼使用。也就是说个人在短暂的时期内使用这个东西，了解自己发生缺陷的地方，并改进自己的习惯。

答：。

							95%区间（工时）		计划工时	累计工时	策划价值	累计策划价值	计划哪一周	实际工时	实际哪一周
	章节	最佳	最可能	最差	Expected	Sigma	下限	上限	Plan Hrs	Cum. Hrs	PV	Cum. PV	Plan Wk	Actual Hr.	Actual Wk.
22.11.7	TDD	2	2.5	3	2.50	0.17	2		4	4	3.1	3.1	1	2	2
22.11.14	MGR领导的任务	4	5	8	5.33	0.67	3		24	28	6.6	9.7	2	7	
	FSP	3	4	5	4.00	0.33	3				5	14.7	2	5.5	
	M3	4	5	6	5.00	0.33	4				6.2	20.9	2		
	M5	4	5	8	5.33	0.67	1.07	20.0	24.3		6.6	27.5	2		
22.11.21										38	66				
	IC	4	5	6	5.00	0.33					6.2	33.7	3		
	CM	8	9	12	9.33	0.67					11.6	45.3	3		
	SiPP	3	4	5	4.00	0.33					5	50.3	3		
	INNOVATE创新	4	5	8	5.33	0.67					6.6	56.9	3		
	M4OCM用Python	2	2.5	3	2.50	0.17					3.1	60	3		
	克服拖延症	2	2.5	3	2.50	0.17	1.36	48.1	53.6		3.1	63.1	3		
	CI	6	8	12	8.33	1.00	1.01	55.8	62.5		10.4	73.5	3		
22.11.28										16	82				
22.12.5										21	103				
22.12.12										38	141				
22.12.19										26	167				
22.12.26										21	188				
23.1.2										38	226				
23.1.9										14	240				