

研发组织该如何设计绩效体系？

德鲁克在《21世纪的管理挑战》一书中指出：“管理的第一个任务是**规定组织的成效和绩效**，而任何有这方面经验的人都可以证明，这实质上是一项最艰巨、最有争议的任务，但同时也是最重要的工作”。知识工作的不确定性更高、组织成员相互协同更复杂，因此，为知识工作者设计绩效更加困难。人力资源同事绞尽脑汁设计指标，但研发管理者和研发团队又觉得指标定义不合理，不断讨论、妥协的结果是，许多企业最后在使用一份有几十个指标的度量体系，管理成本高但效果又不明显。

笔者根据多年企业一线咨询经验，给读者建议了一个研发绩效度量框架作为参考。第一节将介绍这个框架的设计原则，便于读者未来根据企业自身特点对这个度量框架进行定制和调整，第二节将介绍框架中涉及的三种指标类型，第三节将详细介绍这个研发绩效度量框架的具体指标和算法。

研发绩效度量体系设计五原则

1. 外部性

同样在《21世纪的管理挑战》这本书中，德鲁克指出，**任何组织的绩效都只在外部反映出来**。以一个咖啡厅为例，客户等待时间，咖啡口感、价格，这些都是客户可感知的外部指标。研发组织也应优先**选取其客户可感知的外部指标**作为绩效指标，这里客户包括但不限于业务人员，产品经理、最终用户等。

2. 无害性

管理学有一个原则“**你考核什么你就会得到什么**”，绩效指标对团队行为具有很大的牵引作用。设定一个指标后，需要首先考虑负向牵引作用，既团队会采取哪些短期行为来试图迅速达成绩效指标？评估一下这些短期行为对组织的伤害有多大？举个例子，如果用开发代码行数来评估开发工作量，就会对开发工程师产生一个明显的负向牵引作用，让开发工程师失去进行优雅设计或重构代码的动力。因此，绩效度量体系应**尽量选取一些难于伪造或者伪造行为对组织伤害比较小的绩效指标**。

3. 整体性

软件研发组织是一个复杂系统，各岗位间界限很难完全明确。管理者要克制将绩效指标分解到不同岗位，甚至分解到个人的冲动。这种**无效的指标分解往往会导致局部优化**，即各岗位仅仅为自己的绩效而优化工作方法，反而降低了组织的绩效。例如，将进度和质量两个互相牵制的指标，割裂开分配给开发测试，这造成开发只关心进度，忽视移交

质量，而光靠测试来保证质量。这种方式是不可能获得高质量软件的，也不可能达到快速交付的目的。

4. 制衡性

研发组织没法用单一指标来衡量，而需要**用一组指标来互相制衡**以求得平衡。例如，单纯追求交付速度是危险的，必须用质量指标来平衡。

5. 演进性

绩效指标应该随着组织的发展不断调整，需要**定期增减修订指标**，保持指标体系精简，降低管理成本。

三种绩效指标类型

了解了定义研发绩效体系的五个原则后，还需要介绍一下绩效指标的三种类型：

1. 适应性指标

反应组织是否适合生存，这些指标因组织的特点不同而不同。例如，对于一个pizza外卖商家而言，pizza送达时长、送达准时率和价格就是它的适应性指标，但pizza口感就不那么重要。但是，对于一家pizza精品餐厅而言，pizza口感就变得非常重要，而上菜等候时间就变得相对不那么重要了。好消息是，因为研发比较同质化，研发组织适应性指标相对统一。

2. 健康度指标

反映组织的健康程度，就如同一个人的心率、血压，血脂指标一样。健康度指标往往是内部指标，非客户可见，因此需要非常小心衡量指标的无害性和整体性。健康度指标也需要不断调整，一旦一个指标经常保持正常，就可以将它从绩效指标体系中剔除。举个例子，代码冗余度（有多少代码是重复的）就是一个有用的代码健康度指标。

3. 杠杆指标

反映组织的重点改进方向。很多时候，需要进行某项改进，通过杠杆指标来撬动适应性指标改进。例如，今年要推行接口测试自动化，就可以将接口测试代码覆盖率作为一个杠杆指标，这一指标的提升有利于保证生产质量。杠杆指标一般是内部指标，所以同样要保证无害性和整体性。一旦改进活动告一段落，杠杆指标可能会变成健康度指标，也可能直接被剔除。

参考绩效指标体系

有了上面的理论铺垫，可以进入正题，介绍给读者参考的研发绩效度量指标体系。这个体系的指标分成四组，下面将一一详细介绍：

- 1. **响应力**，反映研发组织响应市场要求的能力，包括需求耗费时长，时长分布图K值两个指标；
- 2. **质量**，反映研发组织交付质量，包括生产缺陷需求比，测试缺陷需求比两个指标；
- 3. **可用度**，反映研发组织管理的系统或服务的稳定性，包括系统可用度或服务可用度指标；
- 4. **效能**，反映研发组织的交付效率，包括需求吞吐率，流动效率两个指标。

需求耗费时长

需求耗费时长反映研发组织交付需求的速度。为了计算需求耗费时长指标，需要计算一段时间内所有需求的耗费时长，计算单一需求耗费时长的公式非常简单：

单一需求耗费时长 = 需求上线时间 - 需求提出时间

例如，需求A提出时间为12月5日，需求A上线时间是12月30日，那么，需求A耗费时长就是25天，这里，需求提出时间是指业务人员和研发人员开始澄清需求细节的时间。



在计算出一段时间内，所有单一需求耗费时长之后，可以用以下公式计算需求耗费时长：

需求耗费时长 = 百分位数（所有单一需求耗费时长，85%）

例如，有10个需求，单一需求耗费时间分别是23，42，45，45，45，47，50，62，70，123天，那么取85%百分位数，就相当于取这个数列的第9个数（10*85%后四舍五入得出），最终需求耗费时间是70天，这意味着，85%的需求是在70天内交付的。

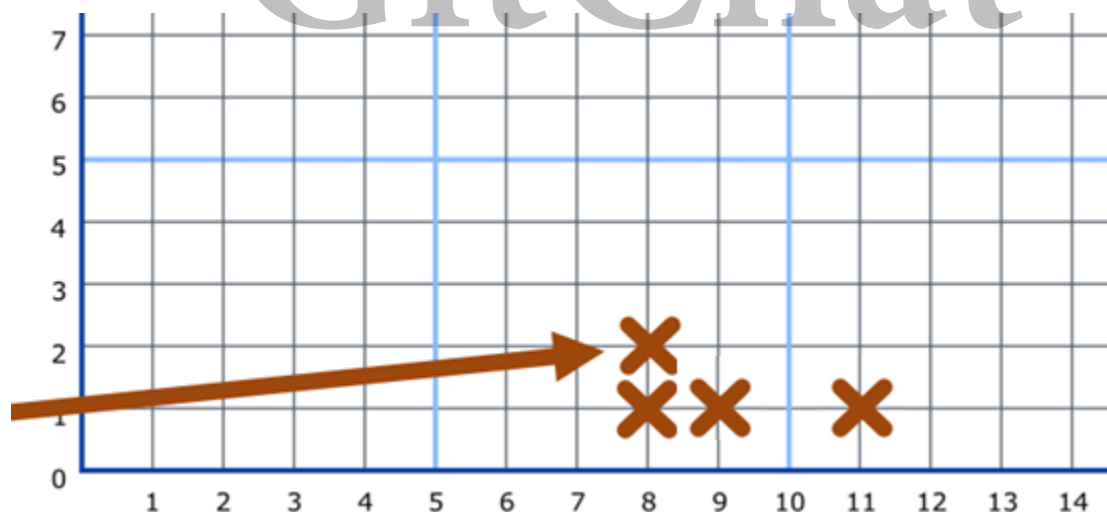
需求耗费时长是一个典型的适应性指标，符合外部性和整体性，研发组织的客户都非常关心，它代表研发组织的快速响应能力。在实际使用中，如果研发组织内存在多个不同需求类型，（常规需求、紧急需求和缺陷）那么就需要将这个指标分成几个不同的指标，如常规需求耗费时长，紧急需求耗费时长以及缺陷修复时长等。

需求耗费时长是反映组织敏捷度的重要指标，因为**需要所有角色齐心协力才能优化**这一指标，所以所有相关角色都应该对这个指标负责，包括业务人员，产品经理，架构师，开发工程师，测试工程师，运维工程师。开发测试及架构师对这个指标的贡献比较容易理解，但产品经理和运维工程师对这个指标也有独特的贡献：产品经理需要保证需求尽量明确，对开发工程师提出的疑问快速响应，对开发完的需求尽快验收；而运维工程师需要在保持系统稳定的前提下，尽量加快移交速度。

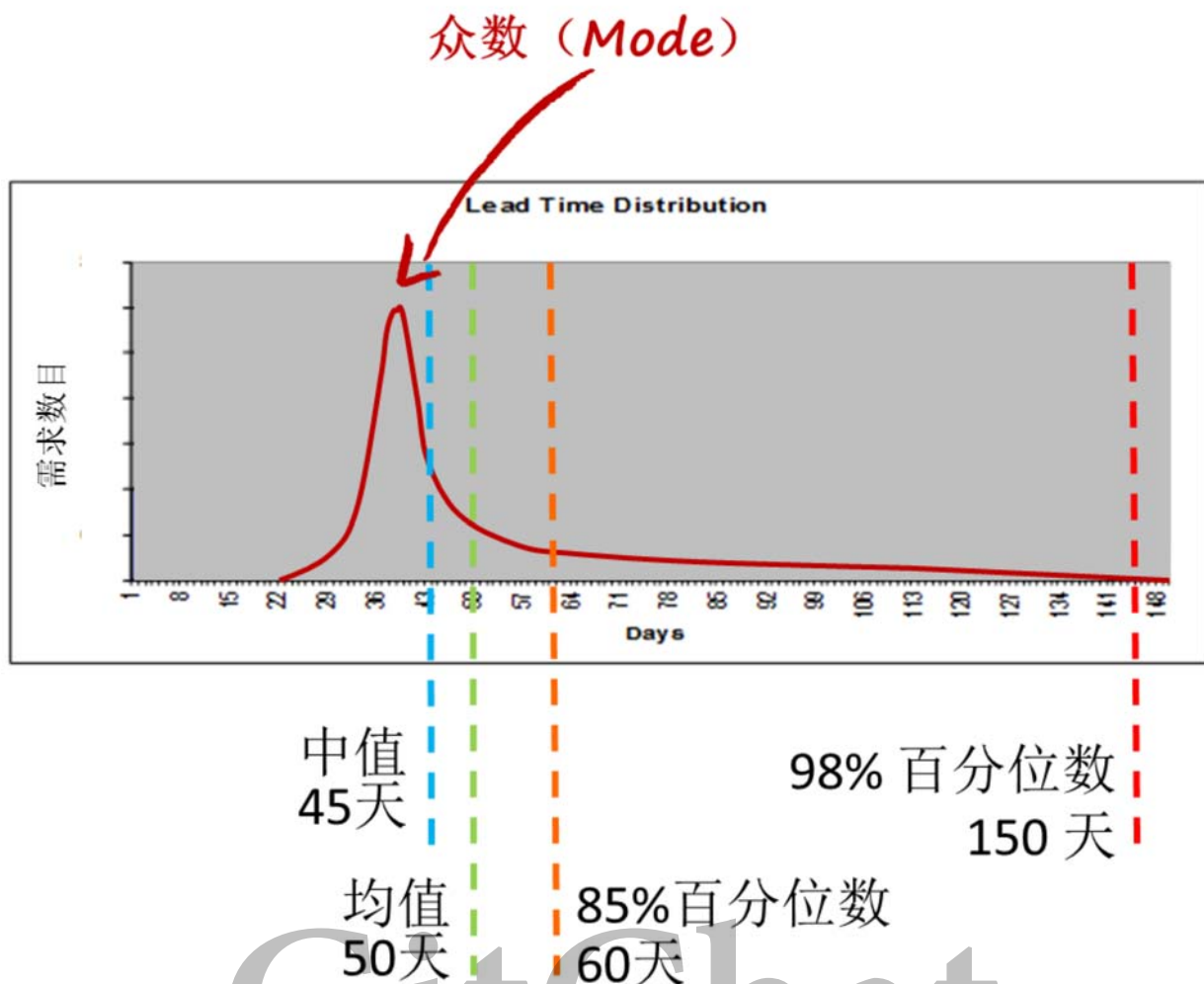
需求耗费时长的一个类似指标是需求完成度，例如2个月需求完成度，就是计算有百分之多少需求可以在2个月内完成。**我不推荐使用需求完成度**这个指标，因为这个百分比指标会引导团队追求100%需求完成度，这不符合软件开发中不确定管理的思路，也忽略了下面一节讨论的需求耗费时间分布分析。

需求耗费时长分布K值

需求耗费时长分布K值反应需求耗费时长的分布特征。为了计算需求耗费时长分布K值，需要先绘制需求耗费时长分布图。分布图是一个柱状图，横轴上X位置柱状高度是需求耗费时长为X天的需求的个数，例如，假设四个需求的耗费时间，分别是8天，8天，9天，11天，那么画出分布图就如下图所示：

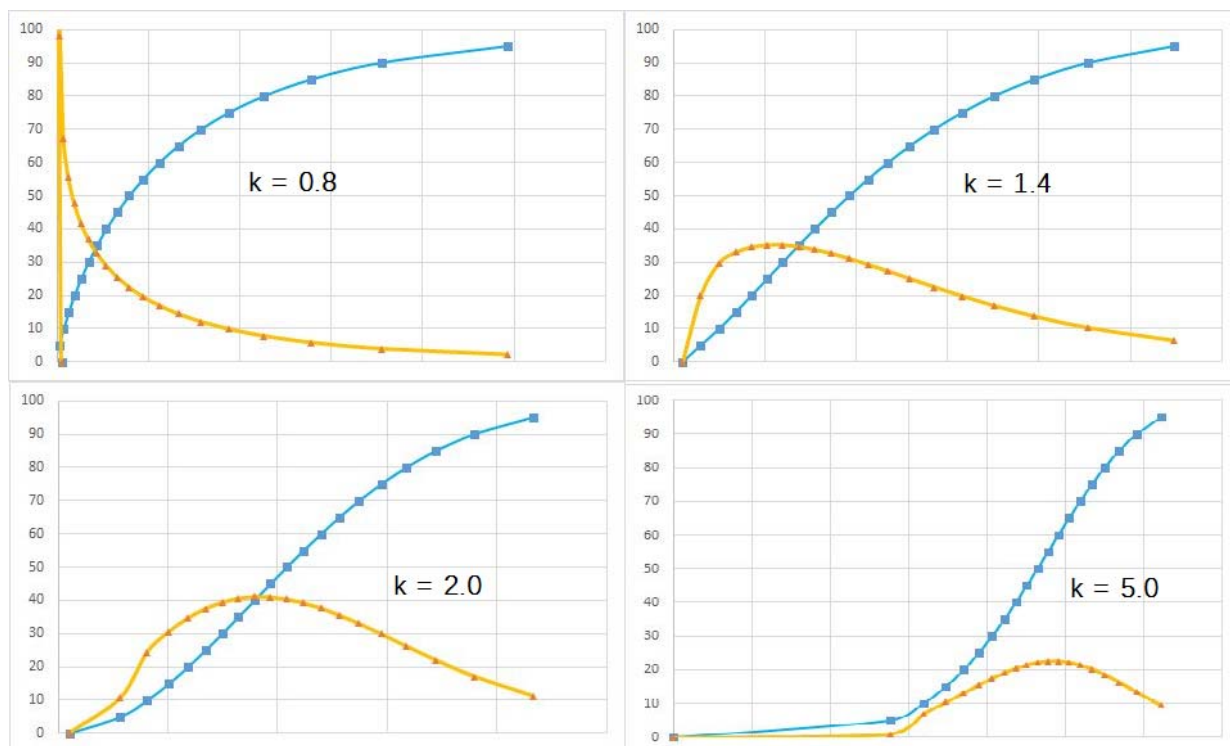


下图是一个实际团队的需求耗费时长分布图，符合韦伯分布（Weibull Distribution），其一个重要特点就是有一个众数尖峰和一个长尾。众数代表大多数需求可以在一个时长区间内交付，是研发系统交付常态；而长尾代表交付系统的意外情况。在下图中可以看到由于长尾的存在，导致需求耗费时长的均值大于中值，85%百分位数大于均值20%，而98%百分位数三倍于均值。



韦伯分布是瑞典工程师韦伯在1930年研究轴承寿命时发现的，他采用了“链式”模型来解释寿命问题。这个模型假设一个结构是由 n 个小元件串联而成，于是可以形象地将结构看成是由 n 个环构成的一条链条，其寿命取决于最薄弱环节的寿命。软件开发可以看成是一个由需求，设计，开发和测试等环节构成的链条，任何一个环节出问题，软件则无法按时交付，这是为什么需求耗费时长也符合韦伯分布的根本原因。

韦伯分布有两个参数，一个是 λ ，一个是 k 。 λ 决定了众数峰值的高度， k 决定了曲线的形状。下图给出了四种 k 值的韦伯分布曲线：



需求耗费时长分布图的K值是一个健康度指标，用来判断需求耗费时长分布是否合理，是否符合可预测性要求。如果K值小于1，那么这个研发交付系统是非常脆弱的，不具备可预测性，需求可能很快交付，也可能会非常慢。如果K值大于2，那么需求交付整体上都很慢，但可预测性比较强；软件开发组织的K值会在1.0到2.0之间。

生产缺陷需求比

生产缺陷需求比反应了研发组织的交付质量。在给定时间内，生产缺陷需求比可以这样计算：

生产缺陷需求比= 生产缺陷数量/上线需求规模

例如，全年生产缺陷200个，上线需求2000个，那么生产缺陷需求比的数值为每需求0.1个缺陷。在实际使用的过程中，如果企业已经有一套生产缺陷分级机制，那么可以使用生产缺陷严重级别对生产缺陷数量进行加权计算。举个例子，假设在200个缺陷里有致命缺陷20个、严重缺陷50个、普通缺陷130个，致命缺陷权值为3，严重缺陷权值为1，普通缺陷权值为0.5，那么加权后的生产缺陷个数是 $(20 \times 3 + 50 \times 1 + 130 \times 0.5)$ ，共175个，加权生产缺陷需求比就是每需求0.0875个缺陷。

使用这个指标的一个挑战是如何确定需求规模。这个首先要看企业是不是已经有一套可行的需求规模估算体系，如功能点，UCP等等。如果有，就可以延续现有的需求规模估算方式。如果没有，那么我强烈建议，在需求上游对需求进行适当拆分，保证需求规模相对均匀，然后**使用需求个数来反映需求规模**。这其中的原因在后面计算需求吞吐率时，会详细解读。

有些企业为了规避需求规模这一难题，使用缺陷移除率这个指标。我不推荐使用这个指标，因为它违反了无害性原则，它会鼓励测试人员多报测试缺陷来提高缺陷移除率，这

样会损害开发测试团队配合。

缺陷移除率=测试发现缺陷数/(测试发现缺陷数+生产缺陷数)

生产缺陷需求比这个指标是另一个重要的适应性指标，不同类型的企业会有不同的要求。它也与需求耗费时长形成了一对制衡，要又快又好才行。生产缺陷需求比应由所有和交付质量有关的人负责，包括产品经理，架构师，开发工程师，测试工程师。

我咨询过程中遇到一些团队，抱怨这个指标已经非常好了，因为生产缺陷都已经私下处理了，没有在系统中记录，因此，还想去寻求别的指标。其实这时不需要纠结，而应该保证生产质量的前提下，将注意力转向下面两个方面:1、进一步缩短需求耗费时长，提高需求交付速度；2.改善开发移交质量，降低测试缺陷需求比，降低测试成本。

测试缺陷需求比

测试缺陷需求比反映开发团队初始移交质量水平。测试缺陷需求比的计算方式和生产缺陷需求比的计算方式类似：

测试缺陷需求比= 测试缺陷数量/移交需求规模

需要注意的是，这个指标应由产品经理、开发工程师和测试工程师来共同负责。他们的目标应该是在尽量在保证生产缺陷需求比的前提下，尽可能降低测试缺陷需求比。例如，去年生产缺陷需求比为0.1个缺陷每需求，测试缺陷需求比为1.5个缺陷，那么今年希望保持生产缺陷需求比先不变，但要把测试缺陷需求比降低到0.5个缺陷每需求。测试缺陷需求比和生产缺陷需求比构成了一对制衡。

测试缺陷需求比是一个杠杆指标，它引导组织的提升其内建质量能力，即由开发工程师在开发过程中同步保证质量，而不是先构建再修复。这背后就是Google一直崇尚的质量观：质量是开发工程师的神圣责任，而不仅仅是测试工程师的责任；只有将开发和测试完全地混合在一起，不分彼此，才能够真正获得好的质量。

测试缺陷需求比的改善会引发一些关联反应，例如，由于测试前置到开发环节参与质量提升活动（如实例化需求，故事验收等工作），一个需求的开发测试时间比会发生变化。Agilean团队之前辅导的一个产品，**测试缺陷需求比下降了90%，同时开发测试时间比从1:1下降到4:1**。另一个长期可能出现变化的指标是开发测试人力比。众所周知，**Google的开发测试工程师比是10:1，而Facebook几乎没有测试**。随着内建质量能力提升，开发测试人力比会逐步提升。

技术债务

技术债务是一个健康度指标，它代表代码内在质量的健康程度，由圈复杂度、代码重复率、每方法代码行等许多指标构成。开源工具SonarCube已经提供了非常好的能力来量化技术债务。

技术债务和需求耗费时间是一对制衡。还技术债需要耗费开发人力，减少当前需求开发人力，短期增加需求耗费时长，但可以优化未来的需求耗费时长。把握好这个权衡主要由架构师和开发工程师来把握。

可用度

可用度指标反映系统或服务的稳定性。可用度的计算公式如下：

$$\text{可用度} = (\text{总可用时间} - \text{不可用时间}) / \text{总可用时间}$$

例如，一个系统的服务水平协议是7天×11小时，那么全年总可用时间就是365×11小时，而假设不可用时间是50小时，那么系统可用度就是98.75%。建议由架构师，开发工程师，测试工程师，运维工程师共同负责改善这个指标，运维工程师主要保证软硬件系统正常，开发工程师和测试工程师主要保证应用系统正常，架构师、开发工程师和运维工程师共同实现DevOps，缩短部署耗时，甚至实现不停机部署。不建议按不可用原因（软硬件系统故障原因，应用故障原因，部署原因）来对这个指标进行细分，这样会增加团队间的摩擦，不利于团队协作。

可用度是一个适应性指标，它适用于自己运维软件、将软件作为服务载体的企业（例如，银行或保险公司）、不适用于将软件作为产品交付出去的公司。这个指标和需求耗费时间是一对制衡，发的版本越多，需求等待越少，需求耗费长越短，但可用度可能越低。

需求吞吐率

需求吞吐率反应研发组织的产能。需求吞吐率就是单位时间内每个开发工程师完成的需求规模，例如每人每月完成两个需求，或者每人每月完成五个功能点。

使用需求个数还是需求估算点数来反映需求规模，主要看企业是否已经有一套成熟且有效运行的估算机制。如果没有，我会强烈建议使用需求个数而不是需求估算点数来反应需求规模。使用估算点数，容易产生两种危害：1. 让研发人员产生规模冲动，想办法（如把需求文档复杂化）来增加估算点数；2. 催生产品经理和研发人员之间的不信任，进而产生讨价还价、合同谈判等浪费，这违反了无害性原则。

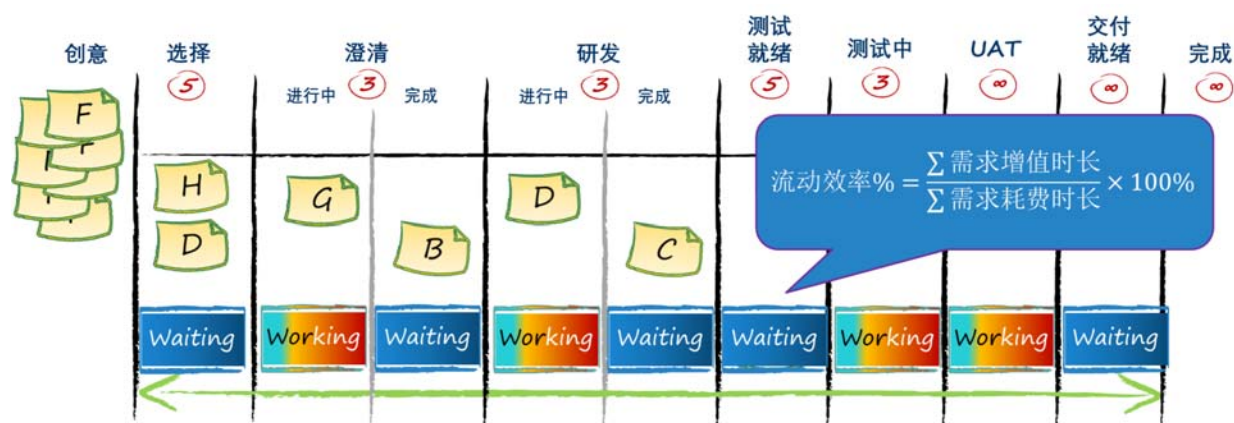
由于上述原因，我建议由产品经理和研发人员一起，将需求分解成颗粒度相对均匀的需求条目，然后用需求个数来表示需求规模。这个指标可能会促使研发人员用动力去更细地拆分需求，但这个副作用对整个组织有利，更小的需求可以更快地交付业务价值。在国际上，用需求个数来替代需求估算的思潮被叫做“**抛弃估算**”运动，如果读者感兴趣可以点击文章末尾的延伸阅读去更多了解。

需求吞吐率是一个适应性指标，它和需求耗费时长形成一对制衡，避免团队单纯改善交付速度，而降低交付数量。但是，研发团队不应该仅仅关注交付需求的数量还应该关心交付后的成效，我建议团队每个人也同时需要对业务指标负责。

流动效率

流动效率反映需求交付过程的效率，流动效率计算公式如下：

$$\text{流动效率 \%} = (\sum \text{需求增值时长}) / (\sum \text{需求耗费时长}) \times 100\%$$



需求耗费时长一节中已经介绍了如何计算需求耗费时长，下面介绍如何计算需求增值时长。计算需求增值时长有三种方式：回忆法、记录法和推算法，下面先用回忆法来说明。访谈参与需求交付的所有角色，请他们回忆交付过程中，他们实际花费了多少个小时，最后将这些时间汇总。假设，需求A澄清环节花了3个人2个小时讨论，研发花了4+6小时，测试花了4小时，改缺陷花了2个小时，UAT花了3个小时，部署花了1个小时，总共花了22个小时，而需求A的交付耗费时长为25天，那么需求A的流动效率就是：

$$\text{需求A流动效率} = 22\text{小时} / (25 \times 8\text{小时}) = 11\%$$

在上述统计过程中，有两点注意事项：

- 建议用小时而不是人天为单位，因为研发人员时间都非常碎片化，使用小时可以促进他们更准确地估算；
- 统计时间，而不是工时，比如3个人花了2个小时澄清需求，只是将2个小时而不是6个小时计入需求增值时长；

了解了如何计算一个需求的效率之后，我们来看如何计算研发交付系统的整体流动效率。例如，需求A增值时长22小时，交付耗费时长为25天，需求B增值时长30小时，交付耗费时长28天，创意C交付耗费时长40小时，交付耗费时长30天，那么整体流动效率是：







$$\text{流动效率} = (22\text{小时} + 30\text{小时} + 40\text{小时}) / [(25 + 28 + 30) \times 8\text{小时}] = 13.8\%$$

流动效率是一个非常重要的杠杆指标，实现自主流动是精益核心思想之一，而**每次成功的精益变革都得益于通过大幅提升流动效率来大幅提升交付时效**，例如，福特用流水线方式生产T型车，将一辆车从12.5个小时缩短到了1小时33分钟；Zara通过集中式生产，集中式上下游整合将服装生命周期从6-9月缩短到最快两周。

上面两个例子都是制造业的例子，但是，软件研发过程中流动效率同样非常低下（有研究表明，**研发全流程流动效率只有1%-5%**），这意味着如果能够对软件研发过程进行精益化改造，同样能够大幅缩短交付时效。软件研发流程的精益化改造虽然复杂程度更高，但同时收益巨大，因此已经成为许多大型企业的研究重点方向。

总结

下表对于那些角色应该关心哪些绩效指标做了一个总结。这个绩效度量体系中的指标可以用于KPI，也可以用于OKR，后面我会再写一篇文章来专门阐述KPI和OKR的区别。

	研发管理	产品经理	交互设计师	架构师	开发工程师	测试工程师	运维工程师
							
需求消耗时长	Y	Y	Y	Y	Y	Y	Y
生产缺陷需求比	Y	Y			Y	Y	
测试缺陷需求比	Y	Y			Y	Y	
技术债务	Y			Y	Y		
系统可用度	Y			Y	Y	Y	Y
需求吞吐率	Y	Y	Y	Y	Y	Y	Y
流动效率	Y						
分布K值	Y						
业务指标	Y	Y	Y	Y	Y	Y	Y

虽然花了许多时间来编写这个指标体系，但是需要声明，**最理想的绩效度量体系还是没有指定度量指标，团队一起齐心协力以业务成功为导向**，这是上策。提出这个指标体系只能算一个中策，至少让团队能够摆脱不合理指标体系的荼毒。

这个指标体系是我提出的企业级精益创新框架（LIFE）的一部分，如果读者感兴趣可以点击下面链接，报名参加“精益创新领导力-研发管理篇公开课”（深圳，3.30-3.31；北京，4.20-4.21；上海，5.25-5.26）。

另外，推荐一门近期的公开课“精益数据分析”，该课程由《精益数据分析》的作者、硅谷大牛Alistair主讲，3月4号在深圳举行。

延伸阅读：

- 《估算三“不”原则》
- 《何时、为何以及如何估算》
- 《估算、切片、剔除》
- 《如何在Scrum中抛弃估算活动》
- 《从康威定律和技术债看研发之痛》