



融·软件项目管理推荐实践库

RRPL手册

Rong Recommended Practice Library For Software Project Management

version 1.0



融·软件项目管理推荐实践库

为软件项目管理的从业人员提供一套系统、完整、落地的推荐实践集

RRPL手册

version 1.0

融·软件项目管理推荐实践库

一、什么是融·软件项目管理推荐实践库？

融·软件项目管理推荐实践库（Rong Recommended Practice Library For Software Project Management, RRPL）是一套面向软件研发领域的项目管理推荐实践库，由国内三家从事项目管理相关咨询、工具开发的资深机构（麦哲思、翰德恩、禅道）联合推出，旨在为软件项目管理的从业人员提供一套系统、完整、落地的推荐实践集。

目前市面上流行的各种项目管理模型或者框架，聚焦的是理论、方法层面，在“道法术器”中的“术”和“器”这一层面并没有涉及。带来的问题就是普通的从业人员很难吃透这些标准方法，在实践执行的时候往往面临无从下手的困境。

因此，融·软件项目管理推荐实践库（RRPL）综合了市面上已知的综合敏捷、DevOps、SAFe、Kanban、CMMI、PMBOK等各种经过验证有用、高效的模型框架体系，并结合国内外几十年来已形成共识的推荐实践，加以系统整理、阐述，形成该推荐实践库。融·软件项目管理推荐实践库可以作为市面上流行的各种模型框架落地执行的指导手册。

在实践库中，我们可以通过角色、模型、阶段等维度来选择适用于自己团队或职能角色的实践；通过“何时使用”以及“如何使用”来确定可以引入团队的实践方法；通过“实践出处”来追本溯源；通过“实战案例”来深入了解实践……对所有的软件项目管理从业者及爱好者来说，可以根据团队的实际情况灵活采用其中的任意实践。

融·软件项目管理推荐实践库希望能够沉淀整个行业的优秀实践，形成即拿即用的项目落地指导手册。

二、融·软件项目管理推荐实践库共创邀请

融·软件项目管理推荐实践库（RRPL）发起者现面向项目管理从业者发出共创邀请，邀您共创RRPL！

自2022年7月1日发布0.1版本至今，RRPL已涵盖需求、设计、开发、测试、部署和发布、运维、度量以及项目管理、团队管理等多个阶段的实践，但我们并不止步于此，接下来将陆续推出项目管理过程中的其他实践。

同时，我们希望融·软件项目管理推荐实践库能够博采众长，欢迎各位小伙伴一起来共创共建，沉淀组织、行业内的优秀实践，搭建系统、完整、注重落地的推荐实践库！在接下来RRPL不断更迭的版本中，一定离不开所有小伙伴的贡献与帮助。

不论是想要沉淀组织内部的优秀实践，还是想要学习和参考其他组织的优秀实践，在共创RRPL的过程中，都能得到意想不到的收获！

三、参与共创能得到什么？

1、得到一个智囊团

RRPL重点关注落地的实践层级，是软件研发的最后一公里，由一个个相对独立的实践组成。RRPL中的这些实践经过各种组合，可以用来应对千变万化的场景！

未来，我们还会围绕这些实践组织各种各样丰富多彩的活动、学习、分享等等，只有想不到，没有我们做不到的！

不管是阅读、学习还是参与共创RRPL，就如同听了一大堆演讲、参加了超多经验交流会，甚至就像是配备了一个参谋部、一个智囊团。在我们的项目管理过程中，这个由众多优秀的实践组成的智囊团能够真正为项目保驾护航。

2、沉淀项目优秀实践

日常项目过程中引入的好玩、有效的实践，可能因为种种原因一直没有形成规范化、条理化的文档，想要分享给他人时却面对的是一团无从下手的线团……

如果有这样的问题，可以在RRPL中将这些优秀的、经过验证的实践整理出来，和我们一起共创、一起交流分享！

3、超多丰富礼品

RRPL推出了全新的积分体系，我们可以作为学习者在融·项目管理平台中注册用户账号，通过注册、每日登录、发帖、回答等行为获取相应积分，详细积分规则如下：

用户行为	积分变动
注册	+50
每日登录	+2
发帖	+5
回帖	+3
参与回答	+5

同时，我们也可以作为贡献者来修改编辑、创建RRPL实践，经审核通过后可以获取相应积分，详细积分规则如下：

用户行为	积分变动
成功创建RRPL实践	+100
成功修改RRPL实践	+10至90分不等，根据重要程度来定
……	……

RRPL通过日排行榜、周排行榜、月排行榜以及总排行榜来实时更新排名，这个排行榜可不是用来“内卷”的。我们会根据排行榜的积分情况，给参与共创RRPL的小伙伴们送上我们的RRPL纪念礼。同时，小伙伴们也可以凭借积分兑换相应的实物礼品，期待在排行榜上看到各位小伙伴的身影！

总排行榜		
排名	姓名	积分
1	Becca	908
2	薛健	648
3	张乐	610
4	郑乔尹	605
5	任甲林	602
6	王连杰	452
7	宋晓娜	434
8	程家乐	434
9	王用涛	424
10	董越	404
11	……	……

四、如何参与共创RRPL？



即刻扫码加入融·实践库共创群。建设属于我们自己的实践库！

目录

需求篇

战略

SWOT分析法	/ 002
发布计划	/ 004
电梯演讲	/ 005
价值主张画布	/ 007
商业模式画布	/ 009
产品路线图	/ 012

需求获取

需求访谈的三驾马车	/ 014
实例化需求	/ 015
用户画像	/ 018
影响地图	/ 020
验收标准	/ 022
用户故事	/ 024
用户故事地图	/ 026
客户旅程地图	/ 028
同理心地图	/ 030

需求排序

MoSCoW排序法	/ 032
RICE排序	/ 034
Kano模型	/ 036
相对权重优先级排序	/ 038
WSJF	/ 040

需求估算

故事点	/ 041
亲和估算	/ 042
计划扑克	/ 044

需求监控

需求跟踪矩阵	/ 046
变更控制委员会	/ 048

解决方案评价

礼宾MVP	/ 049
绿野仙踪MVP	/ 050
着陆页MVP	/ 051
电子邮件MVP	/ 053
碎片拼接MVP	/ 054
众筹MVP	/ 055
单一特性MVP	/ 056
A/B测试	/ 057

开发篇

编码

编码规范	/ 060
结对编程	/ 062
测试驱动开发	/ 064
行为驱动开发	/ 066
集体代码所有权	/ 069
重构	/ 071
探针	/ 073

分支策略

按功能特性创建分支	/ 080
按发布创建分支	/ 082

持续集成

持续集成	/ 084
每日构建	/ 086

代码评审

静态代码分析	/ 075
代码评审	/ 077

测试篇

测试过程

缺陷报告	/ 090
测试计划	/ 092
测试策略	/ 094

测试分类

回归测试	/ 096
系统集成测试	/ 099
用户验收测试	/ 102
单元测试	/ 104
冒烟测试	/ 107

部署和发布篇

发布策略

金丝雀发布	/ 112
蓝绿部署	/ 114
逐特性发布	/ 116
灰度发布	/ 118

发布实施

部署人员参与部署设计流程	/ 120
--------------	-------

运维篇

基础设施服务

创建共享服务	/ 122
--------	-------

人员配备

为每个开发团队分派运维联络人	/ 124
运维工程师融入开发团队	/ 125

度量篇

交付价值

客户问题解决时长	/ 128
客户问题响应时长	/ 129
客户满意率	/ 130
市场占有率	/ 132
用户增长率	/ 133
营收增长率	/ 134
产品价值达成率	/ 135

交付质量

平均故障修复时长	/ 158
平均故障间隔时长	/ 159
线上故障数	/ 160
系统可用性	/ 161
发布成功率	/ 162
测试缺陷密度	/ 163
用例评审通过率	/ 164
用例评审缺陷密度	/ 165
提测成功率	/ 166
代码评审轮数	/ 167
代码走查缺陷密度	/ 168
圈复杂度	/ 169
代码重复率	/ 170
设计评审通过率	/ 171
设计评审缺陷密度	/ 172
需求评审通过率	/ 173
需求评审缺陷密度	/ 174
演示频率	/ 175
测试一次通过率	/ 176
平均服务恢复时间	/ 177
缺陷逃逸率	/ 181
代码覆盖率	/ 184

交付速率

需求颗粒度	/ 137
代码提交频率	/ 138
接口变更率	/ 139
组件复用率	/ 140
组件按时交付率	/ 141
需求变更率	/ 142
需求按时交付率	/ 143
里程碑偏差	/ 144
开发交付周期	/ 145
需求交付周期	/ 146
流动分布	/ 147
流动效率	/ 149
流动时间	/ 151
流动速率	/ 154
迭代速率	/ 156

交付成本

工作量分布

人员流动率

人力成本

技能指数

非人力成本

预算执行率

/ 187

/ 188

/ 189

/ 190

/ 191

/ 192

交付能力

发布时长

发布频率

回滚成功率

部署成功率

部署时长

部署频率

环境整備时长

测试自动化率

/ 193

/ 194

/ 195

/ 196

/ 197

/ 198

/ 199

/ 200

缺陷重开率

构建成功率

构建时长

构建频率

流动负载

/ 201

/ 202

/ 203

/ 204

/ 205

持续改进

过程符合度

审计频率

审计问题关闭率

专项改进完成率

改进效果评价

/ 208

/ 210

/ 211

/ 212

/ 213

综合度量

项目健康指数

/ 214

项目管理篇

整合管理

挣值分析

限制在制品

项目章程

/ 218

/ 220

/ 224

进度管理

时间盒

每日站会

/ 226

/ 228

质量管理

完成的定义

就绪定义

/ 230

/ 234

采购管理

不劳而获，变更免费

滚动合同

/ 237

/ 239

利益相关人管理

干系人登记册

/ 241

团队管理篇

团队组建

同地团队

跨职能团队

/ 245

/ 247

学习工作坊篇

敏捷工作坊

敏捷多米诺骨牌工作坊

/ 252

需求

战略

需求获取

需求排序

需求估算

需求监控

解决方案评价

SWOT分析法

主要贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

优势、劣势、机会和威胁分析（SWOT分析）是一种广泛使用的技术，有助于理解围绕业务需要的高层级观点，并在项目管理的任何级别上比较候选方案。SWOT分析在项目层面变得更详细，在投资组合或项目层面变得更具战略性，并可用于了解组织的优势和劣势以及机会和威胁。



SWOT四象限图

SWOT是四个组成部分的首字母缩写：

- 优势（Strengths）：企业或项目的特点，使其比其他企业或项目更具优势；
- 劣势（Weaknesses）：使企业或项目相对于其他企业或项目处于劣势的特点；
- 机遇（Opportunities）：企业或项目可以利用的环境要素；
- 威胁（Threats）：环境中可能给业务或项目带来麻烦的元素。

实践出处

1965年，斯坦福研究所（Stanford Research Institute）长期规划处的三位同事罗伯特·F·斯图尔特（Robert F.Stewart）、奥蒂斯·J·贝内普（Otis J.Benepe）和阿诺德·米切尔（Arnold Mitchell）撰写了一份技术报告，题为《正式规划：员工规划师在创业中的作用》。该报告描述了担任公司员工规划师的人员如何从评估运营问题的经理那里收集信息，这些问题分为四个部分：当前运营中“满意的部分”，未来运营中的“机会”，当前运营中的“失误”，以及未来运营中的“威胁”。Stewart等人专注于内部运营评估，并将这四个组成部分分为当前（满意和错误）和未来（机会和威胁），而不是像后来在SWOT分析中常见的那样，分为内部（优势和劣势）和外部（机会和威胁）。

同样在1965年，哈佛大学工商管理研究生院的四位同事埃德蒙·P·莱德、C·罗兰·克里斯滕森、肯尼斯·R·安德鲁斯和威廉·D·古思出版了《商业政策：文本和案例》这本教科书的多个版本中的第一个（商业政策是当时流行的一个术语，后来被称为战略管理）。教科书的第一章在没有使用首字母缩写的情况下说明了SWOT的四个组成部分，以及它们在内部和外部评估中的划分。

为什么

使用SWOT分析可以分别从内部和外部、以及正向和负向的角度全方位考虑问题，对方案的选择、取舍、定位、提升竞争力等方面进行高效的决策。

何时使用

SWOT可以在各种层级使用，如在做战略制定、产品定位、特定功能的取舍，以及要做出各种变革之前都可以用SWOT分析来分析利弊，让决策考虑更加周详。

如何使用

1、会议开始前，在一张白纸上写下词语“最终理想状态”，并用一幅图来描绘它。

2、用四张白板纸表示四个不同的方块，按照上下左右的顺序将它们拼起来。如果议题比较复杂或者参与人数比较多，需要更多的方块，那需要多少就用多少。

3、请大家自行静默书写：

a、在左上方的方块中，写下“优势”一词，并画一幅图来描述这个概念。例如，可以简单地画某个人用一只手举着轿车来代表“优势”，是的，允许夸张。留给参与者5~10分钟的时间，让他们安静地思考他们拥有什么强项去实现理想的结果，把想法写在即时贴上，每个想法一张即时贴。

b、在右上方的方块中，写下“劣势”一词，并画一幅图来描述这个概念。让参与者花5~10分钟的时间安静地思考，对于理想的结果，他们的短板在哪里，并将想法写在即时贴上。

c、在左下方的方块中，写下“机遇”一词，同样画一幅图来描述它。留给参与者5~10分钟的时间，在即时贴上写下他们能够想到的机遇。

d、在右下方的方块中，写下“威胁”一词，配图描述这个概念。让参与者利用最后的5~10分钟时间思考面临的危机，写在即时贴上。

4、收集所有想法：

一旦察觉到大家写的动作慢下来了，就可以收集所有的即时贴了，然后把它们贴在相关方块的墙上，让大家都能够看到。一定要确保即时贴是按照主题范围贴的：优势、劣势、机遇、威胁。

5、分类和命名：

为优势、劣势、机遇和威胁四个象限中的便利贴根据相似程度分别进行分类并命名。

6、大家对各个象限中的分类进行“数点投票”。

7、与参与者一同总结在讨论过程中的发现，了解这些发现对于理想状态有何意义。并带领大家开创性地思考问题，鼓励大家以积极的态度评估劣势和威胁，并将这些不良因素作为促成我们实现目标的一种手段。

输出物

SWOT四象限图

参考资料

- 1、https://en.wikipedia.org/wiki/SWOT_analysis
- 2、《游戏风暴》戴夫.格雷桑尼.布朗 詹姆斯.马卡拉佛 著

发布计划

主要贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

发布计划对产品下一次发布（或下几次发布）需要包含哪些功能做出规划。

为什么

发布计划是产品路线图（战略）的战术落地规划，它将战术和战略联系起来。它是产品经理指导开发团队的重要手段，也是管理利益相关人期望的有用工具。

何时使用

发布计划是在制定产品愿景和路线图之后制定的。它用来对产品路线图进行战术层面的规划。

如何使用

发布计划是确定一个或多个主要发布的预期结果并最大化实现它的机会的过程。这涉及以下内容：

- 制定明确、具体和可衡量的目标。描述产品应该创造的结果或好处。我将这些目标称为产品目标。可以在产品路线图上记录它们。
- 确定达成产品目标需要完成哪些工作。这通常需要开发团队提供粗略的、高层级的估算。
- 了解日期和预算限制：是否有任何必须满足的硬性期限，或者预算是否固定？
确定发布日期和发布内容。

注：每个Sprint结束时，都要评审发布计划的进度，并根据需要进行调整。

输出物

发布计划

参考资料

- 1、<https://www.romanpichler.com/blog/release-planning-advice/>
- 2、<https://www.productplan.com/learn/product-roadmap-vs-release-plan/>
- 3、<https://monday.com/blog/rnd/agile-release-planning/>

电梯演讲

贡献人：徐东伟 郑乔尹 晏瑞宇



扫码查看详情

定义

电梯演讲是指在乘坐电梯的时间跨度内（即30秒-2分钟时间内）将想法或产品描述完毕。这个描述要解释产品/想法是什么以及价值是什么。它的目标是以令人兴奋的方式传达整体概念或主题。

实践出处

电梯演讲有很多起源故事。一个众所周知的起源故事是Ilene Rosenzweig和Michael Caruso的故事，这两位前记者活跃于18世纪90年代。那段时间，Rosenzweig正在和Caruso约会。Caruso是《名利场》的高级编辑，当时一直试图向总编辑推销故事创意，但因为这位总编辑永远都在到处走，实在没有大块的时间。因此，Caruso只能在总编辑乘坐电梯去搭乘汽车的时候说上几句。这样在Caruso回到家里的时候，Rosenzweig都会问上一句“你今天的电梯演讲情况如何？”。

Philip Crosby是质量改进行业的先驱之一，他在1981年第2版《获得自己的甜蜜方式的艺术》一书中，建议所有质量改进人员应该有一个预先准备好的演讲，宣传他们新的质量方法的好处，这样在他们无意间发现和公司资深高管同乘一部电梯的时候，就可以向他们推销新想法。但此时“电梯演讲”这个词并没有获得太多关注。1983年，Crosby在英国开设了一个办公室，John Macdonald来运营。John讲授的课程和Crosby在美国讲的课程非常相似，当然向上销售（包括“电梯演讲”）成为课程的核心要素。

在Geoffrey Moore的《跨越鸿沟》这本书中，定义了一个电梯演讲的模板：

- 为了<目标客户>
- 他们不满意<当前市场的替代方案>
- 我们的产品是一个<产品种类>
- 它提供了<令人信服的购买理由>
- 不像<产品替代方案>
- 我们已组装<关键的整体产品功能，能够解决你的特定问题>

为什么

一个想法只有在简短的文字中表述清楚的时候，说明作者是真的想明白了。电梯演讲的格式可以引导作者聚焦于产品最关键的竞争力要素，更高效对产品做出定位。

何时使用

电梯演讲通常在定义产品愿景时使用。

如何使用

1、设定产品的目标，要明确以下几点：

- 产品关键目标。正在开发的这个产品将如何为公司带来利益？这个产品支持公司的哪个特定战略？
- 客户。谁会使用这个产品？
- 需要。客户为什么需要这个产品？对于客户来说，哪个特性最为关键？
- 竞争与差异。这个产品和类似的产品相比较，竞争力如何？有何差异？

2、使用电梯演讲创建愿景声明草案：

- 为了<目标客户>;
- 他们<需要>;
- 这个<产品名称>;
- 是一个<产品类别>;
- 它<产品的好处，购买理由>;
- 不同于<竞争者>;
- 我们的产品<差异/价值主张>。

3、与项目利益相关者、开发团队、Scrum Master、敏捷教练共同审核。

4、确定愿景声明并定期持续维护，确保产品能够符合市场现状并支持公司需求的变化。

输出物

电梯演讲描述

参考资料

- 1、https://en.wikipedia.org/wiki/Elevator_pitch
- 2、<http://www.the-confidant.info/2012/the-history-of-the-elevator-speech/>
- 3、<http://presentingrob.blogspot.com/2016/01/elevator-pitches-brief-history.html>
- 4、<http://www.elevatorpitchessentials.com/essays/CrossingTheChasmElevatorPitchTemplate.html>
- 5、《跨越鸿沟》Geoffrey Moore
- 6.《敏捷项目管理 从入门到精通实战指南》Mark Layton

价值主张画布

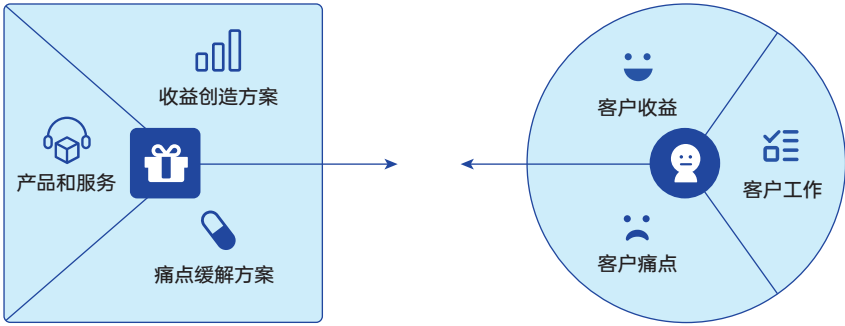
贡献人：徐东伟 郑乔尹 晏瑞宇



扫码查看详情

定义

价值主张画布是从客户的视角理解产品价值的一个工具。它可以帮助了解目标用户的需求，以及产品应该以何种方式满足这些需求。



价值主张画布

价值主张画布由两部分组成，右边为客户细分画布，左边为价值主张模板。这两部分同时也是商业模式画布最核心的两个组成部分。

实践出处

该工具由Alexander Osterwalder、Yves Pigneur和Alan Smith创建。他们同样是商业模式画布的作者，旨在描绘客户所感知的价值。

为什么

价值主张画布有助于：

- 了解客户的需求和期望；
- 根据客户的需求开发产品；
- 将现有产品与用户需求进行比较；
- 找到适合你的产品市场；
- 避免生产没人想要的东西；
- 节省时间和金钱。

总之，通过价值主张画布的绘制，能够帮助分析并提升现有产品/服务与市场的匹配度，或者帮助创造与市场匹配的新产品/服务。

何时使用

- 1、在创业初期。
- 2、为产品添加新功能，这些新功能可能需要时间或资源方面大量的投入。
- 3、要扩展到新市场或客户群，你需要了解这些新客户将如何接受你的产品/服务。

如何使用

1、填写客户细分画布

- 选择客户群：选择你想要描述的客户群。
- 识别客户工作：了解客户正努力完成的工作，在每张便利贴上记下他们的工作。
- 识别客户痛点：客户有什么痛点？把你想到的尽可能多地写在便利贴上，每个痛点一张便利贴。
- 识别客户收益：客户想要获得怎样的结果和效益？把你想到的尽可能多地记下来。每个收益一张便利贴。
- 对工作、痛点、收益进行优先级排序：把最重要的工作、最极端的痛点、最需要的收益放在顶端。

2、填写价值主张模板

- 罗列产品和服务：列出与你现有的价值主张相关的所有产品和服务。每个产品/服务一张便利贴。
- 概述痛点缓解方案：概述你的产品和服务如何通过消除不希望的结果、障碍或风险来缓解客户的痛点。一个便利贴对应一个痛点缓解方案。
- 概述收益创造方案：说明你的产品和服务现在如何对客户创造他们希望或想要得到的结果和效益。一个便利贴对应一个收益创造方案。
- 按重要程度进行排序：对产品和服务、痛点缓解方案以及收益创造方案，根据其客户的必要性进行排列。

3、匹配

- 说明：结合之前你所完成的价值主张图及客户群概况，把痛点缓解方案及收益创造方案逐条通读，看这些是否能与客户工作、痛点及收益相匹配。对每一条已匹配完的条目做好标识。
- 结果：如果痛点缓解方案和收益创造方案未能与任何一项客户工作、痛点和收益相匹配，这表明它将不能给客户创造价值。如果你未能匹配完所有的痛点和收益，别担心，因为你不可能满足所有的要求。扪心自问，你的价值主张与客户的需求契合度有多高？

注意

- 完成价值主张画布只是第一步。之后通过测试和获得反馈来验证假设至关重要。这有助于回到画布，并对其进行优化。
- 还需要强调的是：价值主张画布并不能替代商业模式画布。它们结合起来效果更好。它们不是互斥的。

输出物

价值主张画布

参考资料

- 1、<https://businessmodelanalyst.com/value-proposition-canvas/>
- 2、<https://www.garyfox.co/canvas-models/value-proposition-canvas-guide/>

商业模式画布

贡献人：徐东伟 晏瑞宇 郑乔尹 鲁飞



扫码查看详情

定义

商业模式就是一个描述企业如何赚钱的计划。它解释了你的客户群是谁，你如何为他们提供价值，以及融资的相关细节。

商业模式画布是一种可视化的工具，它允许你把商业模式中九大要素放在同一张纸上，从而更加易于评估商业想法或概念。商业模式画布实际上是另一种形式的商业计划，但它比传统的商业计划更加易用，更加直观，更加高效。



商业模式画布由如下九大部分组成：

1、客户细分（Customer Segment）

你要把产品或服务卖给谁。

2、价值主张（Value Proposition）

为客户提供的利益的集合或组合。

3、渠道通路（Channels）

一家企业如何同它的客户群体达成沟通并建立联系，以向对方传递自身的价值主张。与客户的交流、分销和销售渠道构成了一个企业的客户交互体系。渠道通路在客户体验中扮演着重要角色的客户触点。

4、客户关系（Customer Relationships）

一家企业针对某一个客户群体所建立的客户关系类型。

5、收入来源（Revenue Streams）

企业从每一个客户群体获得的现金收益（须从收益中扣除成本得到利润）。

6、核心资源（Key Resources）

保证一个商业模式顺利运行所需的最重要的资产。核心资源可包括实物资源、金融资源、知识性资源以及人力资源。核心资源可以是自有的，也可以通过租赁获得，或者从重要的合作伙伴处获得。

7、关键业务（Key Activities）

保障其商业模式正常运作所需的最重要的事情。

8、重要伙伴（Key Partnerships）

保证一个商业模式顺利运行所需的供应商和合作伙伴网络。

9、成本结构（Cost Structure）

运营一个商业模式所发生的全部成本。

实践出处

2004年，Alexander Osterwalder介绍了他对商业模式的研究，其中就包含了后来称为“商业模式画布”模板的第一个草图。几年后，画布正式问世，并被确认为分析企业优势和劣势的战略工具。

为什么

- 商业模式画布比传统的商业计划书更加精炼。
- 它是可视化的工具，使任何人都更容易参考和理解。
- 它更易于编辑和共创，可以方便地与员工和利益相关者共享。
- 无论是大公司还是只有少数员工的初创公司都可使用。
- 它阐明了业务的不同方面是如何相互关联的。
- 可以使用商业模式画布模板来引导定义商业模式的头脑风暴会议。

总之，使用商业模式画布，能够使绘制当前商业模式、改进商业模式，以及创造新商业模式这件事情更加简单、浓缩、高效、易用、易懂和直观。

何时使用

当你对未来的道路感到困惑、担忧、不确定、怀疑、犹豫不决时，你就可以使用它。通过商业模式画布将商业模式及其运作方式可视化出来，你就可以在脑子里对下一步如何行动进行上千次的盘算甚至演练。这样你的思路就会变得清晰，可以做出更好的决定。

如何使用

准备工作

- 1、邀请不同角色的人参与，参与者的背景差异越大，描述出来的商业模式越精确。
- 2、手动绘制或打印空白的商业模式画布模板，并贴在墙上。

操作步骤

- 1、先请大家写出企业所服务的细分客户群。不同的客户群使用不同颜色的便利贴，贴在画布上的“客户细分”方格内。
- 2、写出企业针对每个细分客户群提供的价值主张。使用与该客户群相同颜色的便利贴，书写为他们提供的独特价值主张。如果一个价值主张涉及两个差别很大的细分客户群，那么应当分别使用两种颜色的便利贴，分别对应不同的客户群体。
- 3、使用便利贴将商业模式中所有的剩余模块书写出来。记住，针对某一个细分客户群，信息的书写要始终坚持使用同一种颜色的便利贴。
- 4、绘制出整个商业模式后，就可以开始评估该模式的优缺点。可以使用绿色贴点(代表强项)和红色贴点(代表弱项)对商业模式中的各个元素进行评估，分别贴在强项或弱项旁边。
- 5、参与者通过步骤1~4，就可以把组织的商业模式清晰地呈现出来。这时，可以选择对现有商业模式进行改进，或者创建出另外一个全新的模式。理想情况下，需要在新的画布中重新绘制改进版的商业模式或全新的商业模式。

输出物

商业模式画布

参考资料

- 1、https://en.wikipedia.org/wiki/Business_Model_Canvas
- 2、《商业模式新生代》Alexander Osterwalder, Yves Pigneur
- 3、<https://creately.com/blog/diagrams/business-model-canvas-explained/>
- 4、<https://medium.com/@denisoakley/when-to-use-the-business-model-canvas-1601451b2aaf>
- 5、《游戏风暴》

产品路线图

贡献人：徐东伟 许慎 晏瑞宇



扫码查看详情

定义

产品路线图是产品战略目标和期望的高层级可视化概览。它包括产品主要的高层级特性和目标。

为什么

- 有助于描述愿景和战略；
- 为战略的执行提供一个导向性文档；
- 让所有的内部利益相关者能够对齐；
- 有助于和外部的利益相关者（包括客户）进行沟通。

何时使用

制定完产品的战略/价值主张/愿景之后，就可以进行产品路线图的制定。

如何使用

决定你的产品战略。这意味着创建一个详细的计划，在这个计划中：

- 公司确定他们想通过其产品实现什么；
- 并决定一些可能的方法来实现；
- 它还包括对公司买方角色的描述；
- 以及它能给消费者带来的好处；
- 还应该包括你的产品的差异化因素，帮助它在竞争对手中脱颖而出。

这些信息对于想要了解产品所能达到的目标并决定是否应该投资的利益相关者来说是至关重要的。收集关于你的产品的信息。要做到这一点，你需要从不同渠道收集信息。

- 首先，建议与销售和客户支持团队沟通，以获得关于客户的第一手信息。这些部门通常有很多关于客户和他们反馈的信息。增加新的功能或实施新版本的伟大想法可能有助于改善你的产品。
- 第二，与你的社区互动。长期使用你的产品的人肯定知道其优点和缺点。
- 第三，利用你的产品知识。当你知道你的产品的主要特点、优势和弱点时，你可以很容易地想到哪些部件对你的客户最有价值。之后，投入一些时间来消除你的产品可能和现有的缺点。

设定一个时间框架。硬性的最后期限会使团队给出他们无法兑现的承诺。记住，你的产品路线图的主要目的是指导，而不是限制。这就是为什么给团队成员足够的空间和时间来执行必要的任务会更有效率。

为利益相关者准备你的路线图。一个产品往往需要利益相关者的资金帮助才能成功。你可以鼓励你的

合作伙伴在产品开发期间支持你的公司，方法是创建和定制一个路线图，突出利益相关者可能感兴趣的点。不要忘记提到具体的产品功能，你产生销售的潜力，你的产品在竞争对手中脱颖而出的能力，发布日期，好处等。

输出物

产品路线图

参考资料

- 1、<https://www.productplan.com/learn/what-is-a-product-roadmap/>
- 2、<https://sendpulse.com/support/glossary/roadmap>

需求访谈的三驾马车

贡献人：任甲林 徐东伟



扫码查看详情

定义

需求访谈的三驾马车是指在需求访谈时定义三个角色进行需求访谈：

- 主问者：推荐需求工程师担当，事先准备了需求访谈问题单，负责现场对照问题单提问，并根据客户的回答临时追加问题。
- 记录员：可以是需求工程师或开发人员，负责记录客户回答的问题，并思考追加新问题。
- 思考者：推荐测试工程师担当，负责思考需求的可测试性，根据客户的回答追加问题、澄清需求、确保需求可测试。

实践出处

该实践是麦哲思科技在实践中总结的经验。

为什么

通过明确需求访谈的三个关键角色，能够让需求访谈更加全面，可以使团队的工作分工和工作风格保持一致和连贯，确保访谈的质量。同时，针对访谈的改进也因此有章可循。

何时使用

与用户进行需求访谈时使用。

如何使用

通过思考如下的三个问题，进行澄清需求：

- 用户说的名词，要思考What is it? 澄清这个名词是什么意思？是一个概念、实物、还是一个记录信息？是否它还具有属性？如果是代词，指代的是哪个名词？
- 用户说的动词，要思考How to do it? 澄清这个动词是一个功能，还是一个处理逻辑？它具体是如何做的？它处理的数据是什么？它和其他动词有什么关系？它是否还可以细分为多个动作？是否可以举例说明？
- 用户说的修辞语，如形容词、副词，要思考How to test it? 是否可以转化为量词？动词？名词？是否可以举例说明？

变通之法

- 不建议一个人兼任三个角色，太累，访谈效果不好，容易遗漏问题，需要迭代多次。
- 最少2个人，可以主问者同时担当记录员，也可以思考者担当记录员。
- 记录员可以通过录音笔替代之，自动形成文字版记录。
- 主问者可以与思考者每半小时交换一下角色。

实例化需求

贡献人：苗再青 徐东伟



扫码查看详情

定义

实例化需求通俗地讲就是“三剑客”（产品经理、开发人员和测试人员）一起共创需求，最后将需求以实例的方式形成活文档。实例化需求是需求分析的一套流程方法。

在团队进行敏捷转型过程中，如果想提升质量和效率，尤其是质量要想得到明显提升，实例化需求是必须导入的实践。

实践出处

《实例化需求》、在推行实例化需求中的实践。

为什么

- 1、质量提升50%以上，效率提升10%以上，而需要投入的成本只是培训成本（一般是1.5~2个小时）。实践表明，要达到这个效果，1~5个迭代即可；
- 2、有了实例化需求后，开发人员也可以进行测试工作，测试人员更多的进行探索性测试；
- 3、基本上不用再组织PRD评审（PRD是采用实例化需求的形式编写）；
- 4、测试人员的SIT发现的Bug会很少（如果多便将“开发同学按地上狠打一顿，一顿不行咋办？那就再打一顿）。

何时使用

实例化需求方法在需求共创时使用，在Scrum框架中，一般是本迭代对下一个迭代的需求采用实例化需求的方式进行共创。

当团队敏捷进行到高级阶段，可以将实例化需求的共创放到同一个迭代中。

如何使用

1、实例化需求的别称：

以下做法是做的同一件事情：

- 敏捷验收测试
- 实例驱动开发
- 行为驱动开发
- 验收测试驱动开发
- 故事测试
- 实例化需求说明

因此，大家在实践中不要重复导入同一实践，比如说团队有人负责导入了实例化需求，然后又有人听说ATDD（验收测试驱动开发）不错，于是又导入了ATDD。这样会给团队带来混乱，并且会造成重复建设。

当然这有可能和理论派的大佬的观点有冲突，类似于回字的七种写法。你不能说没有细微的差别，但是在实践中如果要体现这种细微，则会付出巨大的成本，有可能会导致失败。

2、实例化需求的别称：

实例化需求的落地实施，一般分两个阶段：

- 初级阶段：手工实例化需求阶段，即实例是采用文档的形式编写，而测试也是手工测试。
- 高级阶段：自动化实例化需求阶段，即采用cucumber等框架工具，实例用Gherkin语言编写，测试是自动化测试。

在开始导入实例化需求的时候，一般都是先从手工阶段开始，很少有一步到位进行自动化测试的。

下面是手工实例化需求的样例（以一个用户故事为例），供参考：

(1) 用户故事：增加业务申请表单

作为（who）：设备管理员

我想（want）：在线上填写零星小型网络的业务申请

以便（why）：流程规范统一可查，及时将需求通知对应供应及区域负责人

(2) 用户故事说明

用户通过点击“业务申请”按钮，发起业务，填写对应的计划使用日期，所属区域，需求相关说明，申请原因（选择所属区域后自动带出对应供应商信息），表单提交之后系统自动发邮件向供应商邀约，现场勘查节点通知到对应区域网络负责人。

(3) 实例说明

客户类型	操作步骤	预期结果
设备管理技师	通过点击业务导航流程图的“业务申请	1. 转到业务申请单据草稿状态； 2. 申请人、部门、单据编辑人、单据修改人默认带出登录人； 3. 单据生成时间、单据修改时间默认带出当前时间； 4. 单据通过时间为空，单据状态为“未提交”。
设备管理技师	点击页面“所属区域”	1. 弹窗显示所有区域信息； 2. 双击选择其中一条自动带出区域负责人、施工位置、负责人电话、负责人邮件、参标供应商等信息。
设备管理技师	点击页面“所属区域”	弹窗显示对应的OU组织信息；双击选择其中一个。
设备管理技师	点击页面“采购类型”、“计划使用日期”	1. 显示所有采购类型； 2. 选择时间，类型仅为日期型。
设备管理技师	输入“申请原因”、“需求说明”，200个字以上，点击“保存”	“申请原因”、“需求说明”红框，并提示：“亲，您输入的文字描述过多，【申请原因】、【需求描述】请控制在200字以下，谢谢！”。
设备管理技师	红色*号不填写内容，点击“保存”。	红色*号字段红框，并提示：“亲，【XXXX】为必填字段，请输入对应内容，谢谢！”
设备管理技师	勾选一个或者多个参标供应商，点击“删除”	对应参标供应商从明细里删除。
设备管理技师	点击“启动流程”按钮。	流程审批通过后，页面区域负责人及对应的参标供应商能收到邀约邮件。

输出物

实例化需求文档（初级阶段）

需求活文档及自动化测试脚本（高级阶段）

参考资料

- 1、《实例化需求》
- 2、《验收测试驱动开发》

实战案例

以下案例来自某大型企业：

- EHS团队导入实例化需求后，在3个迭代内质量提升75%，并稳定在每个迭代平均每个开发人员产生的Bug数在2个以下。
- 花开富贵团队在导入实例化需求之前，团队一个迭代产生的功能性Bug为22个，导入实例化需求后5个迭代，1个迭代产生的Bug数为6个。
- 战狼队，团队开发人员共4人，在导入实例化需求后，连续7个迭代，每个迭代的Bug数不超过13个。

用户画像

贡献人：徐东伟 晏瑞宇 郑乔尹



扫码查看详情

定义

用户画像是基于真实目标用户数据绘制的典型用户的情况介绍，它帮助产品经理（以及组织中参与产品开发的其他人）根据特定类型（而不是广泛意义上的）用户的关键特征、行为、目标、痛点、期待等打造产品。它通常以一张照片（也可以是绘制的人像）搭配相关的信息组成，张贴在人人都方便看到的地方供参考。



姓名：	性别：
年龄：	职业：
描述：	
痛点：	
期待：	

实践出处

在软件设计中，著名的软件开发先驱Alan Cooper提出了用户画像的概念。从1983年开始，他开始使用这个用户画像的原型，使用的数据来自对7-8个用户的非正式采访。从1995年起，他开始研究特定用户而非一般用户如何使用该软件并与之交互。该技术在他1999年的著作《The Inmates are Running the Asylum》中为在线商业和技术社区推广。在本书中，Cooper概述了与创建用户画像有关的一般特点、用途和最佳实践，并建议为单一原型用户设计软件。

为什么

- 使用人物形象而非无形的用户数据，设计师可以更好地共情并推断真实人物可能需要什么。
- 有助于团队对目标用户有共同的理解。
- 防止设计师或开发人员无意识地在产品设计上投射自己的心理模型而造成“自我参照设计”。
- 提升沟通效率和效果，因为更加易懂。

何时使用

考虑品牌购买者和用户的目标、愿望和限制时，帮助指导有关服务、产品或交互的决策，例如网站的功能、交互和视觉设计。

如何使用

1、做调研

首先要收集有关客户的信息，用户画像要基于真实的信息来做。
最好是直接访谈最终用户，实在不行的话，可以访谈与用户接触的角色，如销售及客户服务人员等。

2、做用户细分

分析收集到的调研数据，确定具有相同角色人的共同属性，找到他们的模式。

3、构思用户画像布局

用户画像可以根据产品及团队的实际情况进行个性化定制的。

4、增加人口统计信息

如姓名、性别、年龄、婚姻状态、工作等信息。

5、增加一些额外的背景信息（如有必要）

很多时候，额外的背景信息有助于增强同理心，提供更多的洞察。然而如果提供不恰当，有可能会造成误导。

6、定义用户的目标

定义用户的目标非常重要。这样可以让产品的目标与用户的目标对齐。

7、定义痛点和期待

在用户画像中描述痛点和期待，可以让我们的产品更好赢得用户的忠诚。

判断用户画像实用与否的P-E-R-S-O-N-A

- P代表第一手(Primary research)：指该用户画像是否基于对真实用户的情景访谈；
- E代表同理心(Empathy)：指用户画像中包含姓名、照片和产品相关的描述，该用户角色是否能够引起同理心；
- R代表真实性(Realistic)：指对那些每天与顾客打交道的人来说，用户画像是否看起来像真实人物；
- S代表独特性(Singular)：每个用户是否是独特的，彼此很少有相似性；
- O代表目标性(Objectives)：该用户画像是否包含与产品相关的高层次目标，是否包含关键词来描述该目标；
- N代表数量性(Number)：用户画像的数量是否足够少，以便设计团队能记住每个用户画像的姓名，以及其中的一个主要用户画像；
- A代表应用性(Applicable)：设计团队是否能使用用户画像作为一种实用工具进行设计决策。

输出物

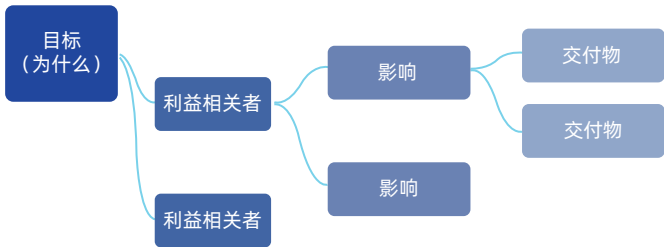
用户画像



扫码查看详情

定义

影响地图是以思维导图的形式，在业务目标和交付物之间建立起强关联。



影响地图分为四个层级：

- 第1层级：目标（Why） - 想要达成的业务目标。
- 第2层级：利益相关者（Who） - 谁能够帮助达成目标？谁会被影响？谁会阻碍目标的达成？
- 第3层级：影响（How） - 你想要让利益相关者有什么样行为的改变，也就是你为了达成业务目标想要对利益相关者产生的影响。
- 第4层级：交付物（What） - 你要做什么（软件交付物）才能产生想要的影响。

实践出处

Gojko Adzic在 2012年自己的新书《影响地图》中将影响地图的概念带给世人的。

为什么

影响地图可以带来如下收益：

- 专注：影响地图只有一个中心节点（业务目标），能够帮助团队保持专注。这大大有助于确定优先级，消除不必要的范围，从而加快交付。
- 在更高层级决策：提供了一种在更高层次上选择和优先排序的方法，如在“影响”层级排序，而不是在故事层级进行优先排序。
- 避免范围蔓延：范围蔓延非常容易发现。不应该成为当前发布周期一部分的用户故事在视觉上不适配影响地图的任何分支。
- 将假设可视化：
 - 假设1：交付物能够对利益相关者行为的变化产生预期的影响。
 - 假设2：利益相关者行为的变化将有助于总体业务目标的实现。

何时使用

使用影响地图的时机包括但不限于：

- 为不存在的产品定义功能；
- 规划下一个迭代或发布，即对特性进行优先级排序；
- 论证某个功能是非常有价值还是毫无价值。

如何使用

如何构建影响地图

- 1、设置业务目标：业务目标要满足SMART。
 - 2、列出利益相关者：列出有助于目标达成的以及有可能阻碍目标达成的利益相关者角色。可以使用用户画像技术来辅助分析。
 - 3、列出影响：为每个利益相关者列出他们行为的哪些变化有助于业务目标的达成。
 - 4、列出交付物：我们要提供哪些功能可以导致/促进利益相关者行为的变化。
- 将相邻层级用线连接起来，表示它们之间的关联关系。

如何使用影响地图

- 1、影响地图是基于假设进行构建的
 - 假设1：交付物能够对利益相关者行为的变化产生预期的影响。
 - 假设2：利益相关者行为的变化将有助于总体业务目标的实现。
 - 我们要选取最有可能有助于目标达成的、最有信心的假设，并结合实施的成本，进行优先级排序逐步实施验证。
- 2、在必要的时候，可以使用客户旅程地图辅助对影响进行进一步的分析。
- 3、用户故事的书写可以参考如下格式
作为<利益相关者>，我想要<交付物>，以便<影响>。

输出物

影响地图

参考资料

- 1、<https://www.impactmapping.org/>
- 2、<https://www.productplan.com/glossary/impact-mapping/>

验收标准

贡献人：徐东伟 晏瑞宇 王连杰 郑乔尹



扫码查看详情

定义

验收标准是需求/用户故事被用户、客户或其他系统接受必须满足的条件，它从最终用户的角度对功能行为进行了定义。每个用户故事都有自己的验收标准。

Mike Cohn将验收标准定义为“关于故事必须做什么才能让产品负责人接受其完成的注释”，它们定义了用户故事的边界，用于确认故事什么时候才叫完成并按预期工作。

在wiki中，验收标准被认为等同于验收测试：

https://en.wikipedia.org/wiki/Acceptance_testing

为什么

- 编写良好的验收标准有助于避免在开发阶段结束时出现意外结果，并确保所有利益相关者和用户对他们得到的结果感到满意。
- 使功能范围更加详细。验收标准定义用户故事的边界。它们提供了有关功能的精确细节，帮助团队了解故事是否已完成并按预期工作。
- 描述异常情景。验收标准定义异常情景，并解释系统必须如何应对它们。
- 建立沟通。验收标准使客户和开发团队的愿景同步。它们确保每个人都对需求有一个共同的理解：开发人员确切地知道该功能必须展示什么样的行为，而利益相关者和客户则理解该功能的预期。
- 简化验收测试。验收标准是验收测试的基础。每个验收标准必须是可独立测试的，因此具有明确的通过或失败场景。
- 进行特性估算。验收标准规定了团队必须开发的具体内容。一旦团队有了精确的需求，他们就可以将用户故事分解为可以正确估算的任务。

总之，不同的人对一个问题可能有不同的观点和解决方案想法，使用明确写出来的验收标准就可以最大程度确保大家对功能做成什么样达成一致。

何时使用

验收标准在需求/用户故事梳理的过程中制定，由产品负责人、开发、测试等人员共同讨论并达成一致。验收标准的确认工作要先于估算活动，因为验收标准不同，工作范围和工作量也会有所不同。

如何使用

1、根据需求的不同，可以用不同的格式编写验收标准，即：

- 面向场景（Given/When/Then模板）；
- 以规则为导向（检查单模板）；
- 自定义格式。

2、面向场景的验收标准：

- 格式
 - 场景：要描述的行为名称；
 - 给定条件（Given）-场景的初始状态；
 - 当（When）-用户做出的具体操作；
 - 就会（Then）-“当”这一步操作的结果；
 - 并且（And）-用来连接并列的部分。
- 例子
 - 场景：忘记密码；
 - 给定条件：用户导航到登录页面；
 - 当：用户选择<忘记密码>选项；
 - 并且：输入有效电子邮件以接收密码恢复链接；
 - 就会：系统将链接发送到输入的电子邮件；
 - 给定条件：用户通过电子邮件接收链接；
 - 当：用户浏览电子邮件中收到的链接；
 - 就会：系统允许用户设置新密码。

3、以规则为导向

- 很多时候不适合用GWT的格式来写验收标准，可以用一组描述系统行为的规则来表达。

- 例子：
 - **用户故事**：作为用户，我想使用搜索字段键入城市、名称或街道，以便找到匹配的酒店选项
 - **验收标准**：
 - 搜索字段位于顶部栏上；
 - 一旦用户单击“搜索”，搜索就会开始；
 - 该字段包含一个带有灰色文本的占位符：“你要去哪里？”
 - 一旦用户开始键入，占位符就会消失；
 - 如果用户键入城市、酒店名称、街道或所有组合，则执行搜索；
 - 搜索语言有英语、法语、德语和乌克兰语；
 - 用户不能键入超过200个符号；
 - 搜索不支持特殊符号（字符）。如果用户键入了特殊符号，则显示警告消息：“搜索输入不能包含特殊符号。”

输出物

验收标准

参考资料

<https://www.altexsoft.com/blog/business/acceptance-criteria-purposes-formats-and-best-practices>

用户故事

贡献人：杜春 郑乔尹 徐东伟 柴颖 晏瑞宇 鲁飞



扫码查看详情

定义

用户故事是需求获取的一种手段：

- 首先，它从用户的视角提供简短的描述，说明想要解决一个什么用户问题；
- 然后，它鼓励大家通过对话来澄清需求的细节；
- 最后，要记录用户故事的细节以及做到什么才算完成的标准。

实践出处

- 1、1997年：Kent Beck 在底特律的克莱斯勒C3项目中介绍了用户故事。
- 2、1998年：Alistair Cockburn访问了C3项目，把用户故事定义为“用户故事是对即将进行的对话做出的承诺”
- 3、1999年：Kent Beck 出版了《极限编程解释》一书的第一版，介绍了极限编程（XP），以及用户故事在计划游戏中的使用。
- 4、2001年：Ron Jeffries 提出了一个用户故事创建的“3C”公式。
 - Card（卡片）：（或者通常是便笺）是一种有形的物理标记，用来保存这些概念；
 - Conversation（对话）：是在利益相关者（客户、用户、开发人员、测试人员等）之间进行的。它是口头的，通常由文件补充；
 - Confirmation（确认）：可以确保对话的目标已经实现。
- 5、2001年：伦敦Connextra的XP团队设计了用户故事格式，并与其他人分享了示例。
- 6、2004年：Mike Cohn 在他的书《用户故事实战》（User stories Applied:For Agile Software Development）中概括了用户故事的原则，这本书现在被认为是该主题的标准参考。
- 7、2014年：Jeff Patton 于2014年发表了用户故事地图技术，该技术旨在通过系统的方法改进用户故事的识别，并通过结构化的方法展示用户故事，以更好地揭示它们之间的依赖关系。

为什么

- 使用用户故事，能够以用户的视角思考问题、收集需求、拆分需求，从而能够相对独立进行开发和上线，尽早向客户交付价值。
- 使用用户故事，鼓励在早期仅提供简短描述，用于进行高阶规划和排序；在临近开发时再澄清细节，这样能够避免过早投入巨大精力，减少浪费。
- 用户故事鼓励通过面对面沟通澄清需求的细节，提升沟通效率和准确性，提升需求收集的质量。

何时使用

只要想到要解决的问题，就可以使用用户故事记录下来（仅提供简要描述即可），放到需求池中，待后期临近实现再澄清需求细节。并在开发的全程根据工作的进展和新的信息更新用户故事。

如何使用

用户故事最常见的格式是：

作为<角色>，我想要<能力>，以便<价值>。

此格式是常用的格式，但不是必需的格式。建议初学者通过这个模板来强制自己养成思考<用户>和<价值>的习惯。

- 当脑子里有任何想法，不管这个想法是否清晰，都可以马上写一个用户故事，放到需求池中。
- 持续对需求池中的用户故事进行优先级排序，优先级高的位于顶端，优先级低的位于底部。
- 对于临近要实现的用户故事进行用户故事的细化和拆分，明确验收标准。
- 如需要，可以对用户故事进行估算，估算单位可以根据实际需要使用人天或者故事点。
- 在研发过程中，可以根据获得的新信息更新用户故事。

对于要临近实现的用户故事，细化拆分时要遵循INVEST原则：

• Independent 独立性

尽可能让用户故事独立于其他用户故事，将需求解耦。否则依赖太强会导致确定优先级、工作量估算都变得困难，而且也无法独立上线。

• Negotiable 可协商性

用户故事并不是在一开始锁定太多细节，而是需要不同的角色在一起沟通，逐渐澄清细节。

• Valuable 有价值

用户故事要有价值，敏捷提倡尽早交付更有价值的软件。

• Estimable 可估算性

用户故事要可估算，不可估算可能是因为故事太大，或者有些信息不够明确，这样不但不利于做计划，还会存在很多的风险。

• Small 短小

要确保用户故事足够小，最低限度是要在一个迭代中能够完成。通常建议用户故事可以在半个迭代时间长度内完成。

• Testable 可测试性

用户故事要有验收标准，明确说明用户故事做到什么程度才算是完成。

输出物

用户故事

参考资料

https://en.wikipedia.org/wiki/User_story

用户故事地图

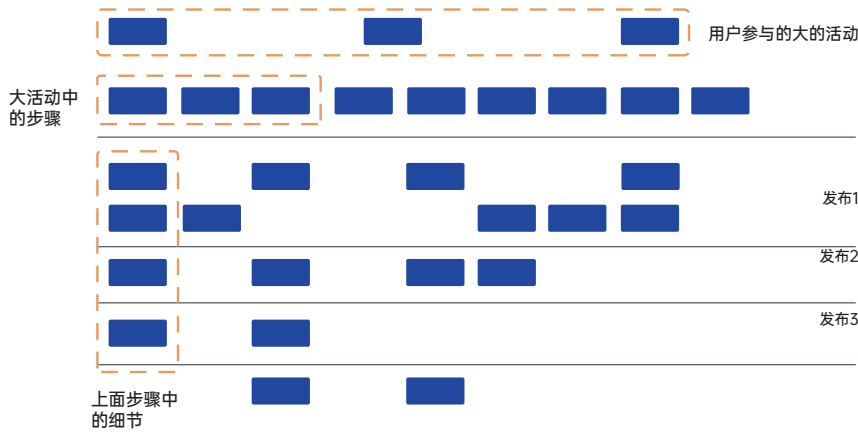
贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

用户故事地图是一种安排用户故事的方法，将客户旅程的基本步骤安排在水平轴（行）上，将用户故事安排在相应的步骤（列）下面，在同一列中，用户故事的优先级由上至下依次降低。当用户故事地图完成时，可以在单一的逻辑视图中看到用户与产品交互的所有方式，从第一次交互到完成总体用户目标。使用用户故事地图，可以以更全局的视角了解用户故事如何融入整体用户体验。



实践出处

- Jeff Patton发明的用户故事地图，这个想法来自交互设计中使用的客户旅程地图。
- 2005年：Jeff Patton在《It's All in How You Slice It》中阐述了用户故事地图的概念，但没有给它起这个名字。
- 2008年：Jeff Patton的《The new user story backlog is a map》对用户故事地图实践进行了描述和充分说明。

为什么

1、用户故事地图将软件交付成果与客户旅程和业务流程联系起来，展示了各个故事如何为更大的全局做出贡献，并为发布计划提供了良好的视觉呈现。

2、用户故事地图是一个很好的可视化工具：

- 可以帮助交付团队让业务利益相关者参与进来，一次性对需要、潜在解决方案和交付计划进行富有成效的讨论。
- 简单的表格结构易于理解，对业务利益相关者来说也很自然。
- 卡片垂直摆放允许团队使用不同的发布方案，并在承诺解决方案之前探索不同的选项。

3. 用户故事地图的优势：

- 看见全局，理解为什么做软件。用户故事很好，因为它们帮助交付团队从用户的视角考虑问题。故事地图更进一步，将故事放在更广泛的用户旅程的角度。它们帮助团队更好地了解他们为什么要构建软件，以及软件如何融入全局。
- 场景化，考虑问题更周全。除了一些罕见的例外，任何软件产品都只是更广泛的业务活动或客户体验的一部分，

通常在与软件交互之外开始和结束。软件可能只是加速某些活动的某一部分，或者提供一种新的做事方式。规划用户旅程有助于团队在更广泛的背景下考虑用户故事，发现遗漏或不必要的步骤，并创造新的产品创意。

- 避免线性思维。以表格的方式组织故事也有助于避免线性思维，支持待办事项分层。正因为如此，故事地图有助于优先级划分、故事拆分，并为发布计划提供重点。特别是，通过规划一段旅程，然后在每个步骤中考虑故事，这样故事地图帮助团队将用户故事视为选项，而不是承诺。
- 将交互设计融入迭代交付。故事地图提供了一个很好的框架，用于考虑用户交互过程，并在此背景下绘制迭代版本和里程碑。这样就更容易规划交互设计工作，优先考虑与即将到来的里程碑相关的部分。
- 促进迭代交付。通过将相关故事分组在一起，利益相关者通常会看到，他们可以通过更简单的可交付成果选择来启用某些操作，并将更复杂的故事推迟到以后的版本中。

总之，使用用户故事地图，就可以“既见树木，又见森林”，考虑问题更全面，规划交付更轻松，并引导以用户的视角看待问题，提升软件的价值，同时提升与用户沟通的效率。

何时使用

用户故事地图可以在产品开发过程中的任何时候使用，以推动讨论并协调团队。可以为新产品绘制用户故事地图，也可以为现有产品绘制。一旦创建，团队将随着时间的推移维护和引用用户故事地图；他们对其进行添加、修改以反映产品的实际状态，并使用它来指导后续工作和要发布的内容。

如何使用

步骤

- 1、确定想要为哪个客户细分或用户画像制作用户故事地图。
- 2、确定故事地图的主干——水平轴。确定关键的高层级用户活动（而不是软件解决方案），如需要，将用户活动分解为高层级步骤，放在用户活动的下面一层，这些步骤也不应暗示特定的技术或解决方案。
- 3、一旦主干就位，尝试把各个步骤的各种选项写成用户故事，从上到下，优先级由高到低排列。
- 4、通过垂直移动用户故事就可以计划发布。

注意

- 1、使用物理卡片或便签在大白板或墙上创建故事地图，以便轻松移动它们。过早地将地图放入软件工具会扼杀对话。物理卡片允许利益相关者轻松地进行垂直定位和优先级排序，并讨论潜在的迭代发布。
- 2、如果一个用户细分有2-3次可能涉及你的系统的旅程，那么通常最好创建一个组合故事地图，这些旅程在主干上从左到右排列。这将为你提供全局概览，并帮助你对里程碑进行高层级的优先排序。对于更复杂的情况，为每个旅程制定单独的地图更好。
- 3、如果你的产品有很多用户细分，他们都有潜在的旅程，可以考虑使用影响地图来设定愿景和高层级优先级。每个影响都可能是一个用户旅程中的一个变化，因此你可以为最重要的影响创建一个用户故事地图。

输出物

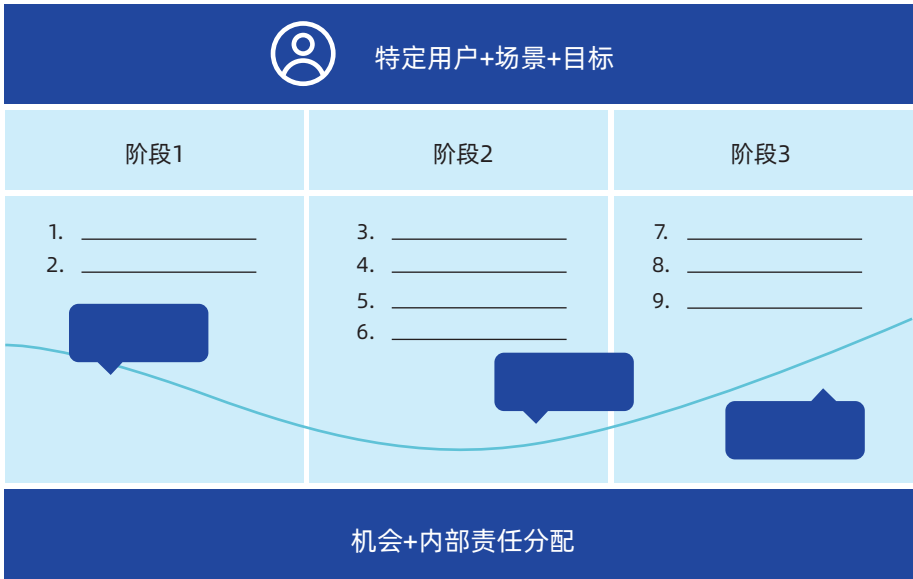
用户故事地图

参考资料

- 1、《用户故事地图》Jeff Patton 著
- 2、<https://www.agilealliance.org/glossary/storymap/>
- 3、http://www.jpattonassociates.com/wp-content/uploads/2015/01/how_you_slice_it.pdf
- 4、<https://www.jpattonassociates.com/the-new-backlog/>
- 5、<https://www.productplan.com/glossary/story-mapping/>
- 6、<https://www.nngroup.com/articles/user-story-mapping/>

定义

客户旅程地图描述了单个用户为了实现其目标而必须执行的阶段和行动步骤、客户在整个过程中的情绪变化、不爽的触点，或未满足的需求等。客户旅程地图用于改进产品的设计，允许用户参与其中。



客户旅程地图的表现形式不限，并且包含的信息元素除上面提及的以外，还可以增加其他想要的元素。

实践出处

客户旅程地图一词的确切来源尚不清楚。不过直到客户体验管理领域在世纪之交成为人们关注的焦点，旅程地图才出现。它是从服务蓝图衍生而来的。

为什么

- 旅程地图让组织将注意力放在客户的想法、行为以及情绪上，组织能够看到通常知之甚少的真实的用户体验。引导大家的视角从“由内向外”转变为“由外向内”。
- 因为旅程地图构建的是完整的客户旅程，所以旅程地图成为跨部门对话和协作的工具。
- 旅程地图通过识别出不爽的触点，帮助找到优化用户体验的起点。
- 可以有助于解释一些量化数据所反映出的问题。

何时使用

如果你想知道要把用户体验的改进重点放在哪里，最好使用客户旅程地图来识别；如果你知道问题在哪里，你只需要找出解决方案。

如何使用

- 1、选择为哪个角色描绘旅程地图。如果有多个不同的角色，要绘制多张旅程地图。
- 2、说明场景和期待：
 - 场景是该旅程地图想要解决的问题域的综述。
 - 期待是角色对解决方案域的期待。
- 3、确定高层级阶段，方便问题的分析。
- 4、填写动作、心态和情绪：
 - 动作：角色在相应阶段采取的步骤。
 - 心态：角色在旅程的不同阶段中各个触点的想法、问题、动机和信息需求等。理想情况下这来自于客户访谈的逐字记录：可以结合同理心地图使用。
 - 情绪：横跨各个阶段的单条线，表示角色的心情在各个阶段的心情高涨还是低落。
- 5、机会：
 - 是从整个旅程地图中获得的洞见，说明如何优化用户体验。
 - 包括我们需要做些什么来改善？谁负责哪些修改？最大的机会是什么？我们如何度量这些改进？
 - 客户旅程地图应该是动态维护的，随着时间的推移和更多信息的获得，要持续更新客户旅程地图。

输出物

客户旅程地图

参考资料

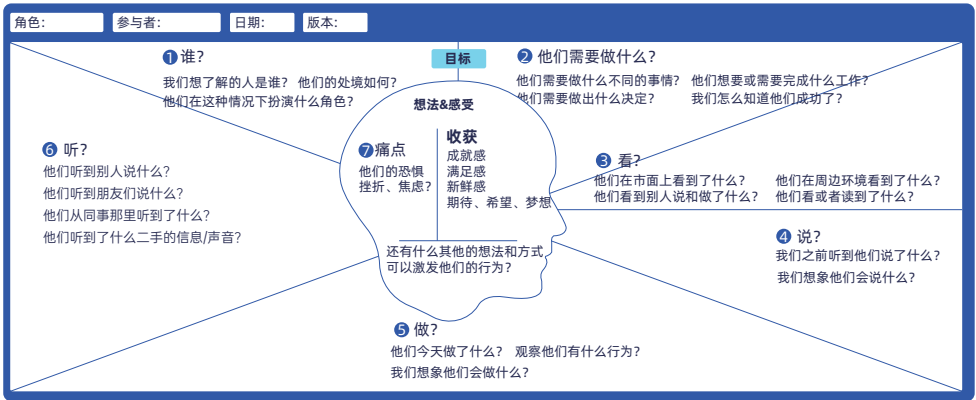
- 1、<https://www.nngroup.com/articles/journey-mapping-101/>
- 2、<https://www.nngroup.com/articles/customer-journey-mapping/>
- 3、<https://www.cleverism.com/everything-you-need-to-know-about-customer-journey-mapping/>
- 4、<https://www.freestyle.agency/blog/customer-journey-maps-and-customer-experience-maps-the-why-when-what/>
- 5、<https://www.interaction-design.org/literature/topics/customer-journey-map>



扫码查看详情

定义

同理心地图是一个可视化工具，它能够帮助团队使用同理心来获取对于目标群体的洞察。



该工具提供了一系列的提示，以确定目标群体的想法、感受、动机、期待和需要。这迫使调研的团队关注目标群体的需求，而不是自己的需求。

实践出处

同理心地图由Dave Gray开发，他是战略咨询公司XPlane的联合创始人，也是《游戏风暴》的作者之一。

为什么

在任何用户体验设计的过程中都应使用同理心地图，这样可以在团队成员之间建立共识，以及了解用户需求并确定其优先级。在以用户为中心的设计中，最好从设计过程的一开始就使用同理心地图。

无论是制作同理心地图的过程，还是最后完成的地图本身，都对组织有重要的好处：

- 刻画用户或角色。同理心地图的制作过程有助于将您对用户知识提取和分类到一个地方。它可用于：
 - 能够对之前所做的定性调研结果（如调研笔记、问卷答案、用户访谈记录）进行分类和理解。
 - 发现您当前知识中的差距，并确定解决这些差距需要那种类型的调研方法。如果同理心地图看起来很稀疏，说明需要进行更多的调研。
 - 通过对同理心地图进行对齐和分组，可以创建用户画像。
- 与他人交流用户或角色：同理心地图是一种快速、易懂的方式来说明用户的态度和行为。一旦创建，它应该在整个项目中充当真相的来源，并保护它不受偏见或毫无根据的假设的影响。

总之，同理心地图通过“设身处地”为客户着想，可以确保他们的需求和愿望得到了充分考虑。

何时使用

同理心地图可以在如下场景中使用：

- 商业模式设计。如果你使用同理心地图来帮助设计你的商业模式，你可以从一开始就获得巨大的优势。
- 产品开发。帮助开发者/工程师与那些实际使用你的产品的人产生共鸣。
- 开发客户资料。虽然客户资料是用户画像开发的一部分，但同理心地图可以帮助团队成员为这些用户画像添加更多深度。
- 用户界面和用户体验讨论。这种讨论有助于设计师和开发人员做好他们的工作，但所有部门的人员都应该参与其中。
- 了解用户行为。当你发布了一个新产品，并在使用和可能的改进方面度量其影响。

如何使用

1、确定主题和同理心地图的范围

- 确定要为谁做同理心地图。可以使用用户画像来展示目标群体。
- 做同理心地图要达成什么目的。可事先使用客户旅程地图找到想要用同理心地图考察的“接触点”。

2、收集相关数据

- 同理心地图基于实际的信息（如想法、感觉或者视角）来做效果最好。
- 通过访谈、调查问卷、观察或者听销售电话。
- 掌握良好的问问题技巧获取想要的信息，如果是面对面交谈，使用积极聆听并注意非语言表达。
- 如果不能直接做访谈，市场细分技术也可以帮助定义目标群体和他们的需要。
- 确保同理心地图制作的参与者要提前阅读事先收集到的数据，理解本次调研的主题和范围。

3、开始填写同理心地图

- 从图的右半部分第3步“他们看到了什么？”开始。根据之前收集到的数据和反馈回答上面的问题。
- 如果可能，每个人要至少在这个部分添加一个评论。

4、完成同理心地图的外圈所有部分

- 顺时针填写，与第3步的方法类似。
- 尽可能使用收集到的目标群体的语言，而不是总结和解释。

5、完成同理心地图的核心区域

- “痛点”是他们感到焦虑、受挫或生气的东西。
- “收获”是他们想要、需要或者希望有的。
- 参考从第2步收集到的信息。它们展示了利益相关者想什么以及感受如何？让团队从客户的角度思考问题。

6、反思到目前为止的发现

- 整个团队退后一步看看大家做出来的同理心地图。并询问大家的想法和建议。

输出物

同理心地图

参考资料

- 1、<https://www.mindtools.com/pages/article/empathy-mapping.htm>
- 2、<https://www.searchenginepeople.com/blog/empathy-map.html>
- 3、<https://www.nngroup.com/articles/empathy-mapping/>

MoSCoW排序法

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

MoSCoW方法是一种用于管理、业务分析、项目管理和软件开发的优先级排序技术，用于与利益相关者就需求的重要性达成共识。

MoSCoW这个词本身是一个首字母缩略词，来源于四个优先类别的第一个字母：M-Must have(必须有)，S-Should have(应该有)，C-Could have(可以有)，Won't have this time(这次不会有)。

MoSCoW经常与时间盒(Timeboxing)一起使用，在时间盒中，实践期限是固定的，因此必须将重点放在最重要的需求上。MoSCoW通常用于敏捷软件开发方法，如Scrum、快速应用程序开发(RAD)和DSDM。

实践出处

这种优先级排序方法由Dai Clegg于1994年开发，用于快速应用程序开发（RAD）。从2002年开始，MoSCoW与动态系统开发方法（DSDM）一起广泛使用。

为什么

使用MoSCoW优先级排序方法，能够很方便地将需求大体分为四类，在有限的时间和预算情况下选取更高优先级的工作来做。

何时使用

在任何需要对需求进行排序或者做计划的时候都可以选择使用MoSCoW优先级排序方法，如做需求梳理、发布计划、迭代计划等等。

如何使用

理解了如下MoSCoW的详细解释，就可以在实际的工作中根据需要灵活使用：

- 必须有（Must have）：标记为“一定有”的需求对当前交付时间盒的成功至关重要。如果哪怕是有一个“必须有”的需求没有包括在内，那么项目交付应该被视为失败（注：与所有相关利益相关者达成协议后，可以将需求从“必须有”降级；例如，当新的需求被认为更重要时）。
- 应该有（Should have）：标记为“应该有”的需求在当前交付时间盒中很重要，但并非必要。虽然“应该有”的需求可能和“必须有”的要求一样重要，但它们往往在时间上没有那么要紧，或者暂时可以使用另一种方法来满足需求，这样就可以推迟到未来的交付时间盒。
- 可以有（Could have）：标记为“可以有”的需求是期望有的，但不是必需的，可以通过少量开发成本来改善用户体验或客户满意度。如果时间和资源允许，这些通常会包括在内。
- 不会有（Won't have）（这次）：被标为“不会有”的需求，已被利益相关者视为最不关键、最低回报的项目，或者在当时不合适。因此，“不会有”的需求不会被计划接下来的一个交付时间盒中。将“不会有”的任何需求删除或重新考虑包含在以后的时间盒中。

输出物

各项工作的MoSCoW属性

RICE排序

贡献人：徐东伟 龚敏睿 晏瑞宇



扫码查看详情

定义

RICE是用来对优先级进行评估的四个因素的开头字母缩写：影响范围（Reach）、影响力（Impact）、信心（Confidence）和工作量（Effort）。RICE值=（影响范围X影响力X信心）/工作量，值越大，优先级越高。

实践出处

RICE模型由Intercom公司开发，Intercom公司一直在使用评分系统对不同的想法进行优先级排序，但他们找不到最适合自己的。经过多次试验和打磨，他们提出了RICE排序方法，将影响范围（Reach）、影响力（Impact）、信心（Confidence）和工作量（Effort）这四个因素结合起来对工作进行排序。

为什么

RICE排序公式输出一个分数，即使待排序的工作类型差异很大，也可以一致地应用该分数，为团队提供一种客观的方法，确定在其产品路线图上优先顺序。

何时使用

RICE优先级模型可用作产品路线图规划会议的输入数据。

如何使用

RICE排序的四个因素详细解释如下：

- 影响范围（Reach）：在规定的时间内估计有多少人会受到影响？（在规定的时间内进行估计）每个团队可自行决定“规定的时间”（如1个月或3个月）和“多少人”的单位定义（如是访问人数还是交易数量）评估影响范围（Reach），既可以使用绝对值（影响用户数），也可以使用更为简单的相对值（受影响比例），如：
 - 10.0 = 巨大影响（超80%的用户会受到影响）；
 - 6.0 = 大面积影响（50%~80%用户会受到影响）；
 - 3.0 = 显著影响（25%~50%用户会受到影响）；
 - 1.5 = 小范围影响（5%~25%用户会受到影响）；
 - 0.5 = 极小的影响（少于5%）。

- 影响力（Impact）：这会产生多大的影响？影响可能指增加收入、降低成本或风险、增加品牌或收益机会等方面。

巨大 = 3，高 = 2，中等 = 1，低 = 0.5，最小 = 0.25

- 信心（Confidence）：你对自己的估计有多大信心？信心包括有多了解用户痛点，对解决方案和技术细节有多大程度的把握等。

高 = 100%，中 = 80%，低 = 50%

- 工作量（Effort）：这需要多少个“人月”？（使用整数和最少半个月的时间——不要掉进估算的坑里。）

RICE值 = （影响范围X影响力X信心）/工作量，值越大，优先级越高。

当然，RICE分数不应该成为一个硬性规定。在很多情况下，你可能首先要做一个分数较低的工作，比如当这个工作是另外一个工作的依赖时。

输出物

各项工作的RICE值

参考资料

- 1、<https://www.intercom.com/blog/rice-simple-prioritization-for-product-managers/>
- 2、<https://chisellabs.com/blog/rice-scoring-model-a-prioritization-framework-overview-for-product-managers>
- 3、<https://tldv.io/blog/rice-prioritization-framework/>

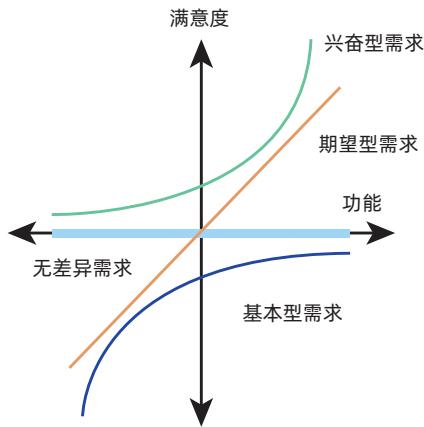
Kano模型

贡献人：徐东伟 晏瑞宇



定义

Kano模型是东京理工大学教授狩野纪昭（Noriaki Kano）发明的对用户需求和排序的有用工具。通过分析用户对产品功能的满意程度，以及对产品功能进行分级，来确定产品实现的优先级。



Kano模型有两个维度：

- 第一个维度是“客户满意度”，表示客户对实现（或不实现）特定特性的满意度。
- 第二个维度是“功能”，它表示该特性在多大程度上实现了预期的功能。

Kano模型将潜在特性分为四类：

- 兴奋型需求：若不提供此需求，用户满意度不会降低；若提供此需求，用户满意度会有很大的提升。
- 期望型需求：当提供此需求，用户满意度会提升；当不提供此需求，用户满意度会降低。
- 基本型需求：当不提供此需求，用户满意度会大幅降低，但优化此需求，用户满意度不会得到显著提升。
- 无差异需求：用户根本不在意的需求，对用户体验毫无影响。

随着时间的发展，特性会在Kano模型中向下移动。即原来的兴奋型需求有可能会慢慢变为期望型需求，继而变成基本型需求。

实践出处

受行为科学家赫兹伯格的双因素理论的启发，东京理工大学教授狩野纪昭(Noriaki Kano)和他的同事Fumio Takahashi于1979年10月发表了《质量的保健因素和激励因素》(Motivator and Hygiene Factor in Quality)一文，第一次将满意与不满意标准引入质量管理领域，并于1982年日本质量管理大会第12届年会上宣读了《魅力质量与必备质量》(Attractive Quality and Must-be Quality)的研究报告。该论文于1984年1月18日正式发表在日本质量管理学会(JSQC)的杂志《质量》总第14期上，标志着狩野模式(Kano model)的确立和魅力质量理论的成熟。这篇论文中，狩野纪昭在首次提出满意度的二维模式，构建出Kano模型。在当时的日本，提高产品和服务这个问题一直都是一个难题。他提出的这个模型有效的解决了这个问题。

为什么

Kano模型能够帮助识别用户对潜在新功能的接受度，从而对待开发的功能按照对用户满意度的潜在影响进行排序，践行“以用户为中心”的价值观。

何时使用

在制定产品路线图或者做发布计划时，用于确定要有哪些需求或者决定需求要做到什么程度的时候使用。

如何使用

向20-30个用户进行一次书面的问卷调查，对于每个特性，都问两个相反的问题，每个问题都设置5个不同程度的可选项，如下：

如果你不能上传照片，你觉得怎么样？	非常喜欢		如果你可以上传照片，你觉得怎么样？	非常喜欢	×
	理应如此			理应如此	
	无所谓			无所谓	
	勉强接受			勉强接受	
	很不喜欢	×		很不喜欢	

针对每个特性，参考收到的答案，根据下表，得出对于每个用户来说，每个特性属于什么类别（字母M/R/L/Q/E/I）。

客户要求		功能缺失问题				
		非常喜欢	理应如此	无所谓	勉强接受	很不喜欢
功能存在问题	非常喜欢	Q	E	E	E	L
	理应如此	R	I	I	I	M
	无所谓	R	I	I	I	M
	勉强接受	R	I	I	I	M
	很不喜欢	R	R	R	R	Q

• M - 基本型 • R - 反向型 • L - 期望型 • Q - 存在疑问的 • E - 兴奋型 • I - 无差异

为每个特性，计算出它在所有受访用户心中所属类别的分布情况。分布最集中的类别，即是该特性针对所有受访用户这一群体的核心类别。

主题	E	L	M	I	R	Q	类别
对项目成绩作图	18.4	43.8	22.8	12.8	1.7	0.5	期望型
上传照片	8.3	30.9	54.3	4.2	1.4	0.9	基本型
发布简历	39.1	14.8	36.6	8.2	0.2	1.1	兴奋型基本型

参考资料

- 1、https://en.wikipedia.org/wiki/Noriaki_Kano
- 2、<https://www.quora.com/Why-is-the-Kano-model-important>
- 3、<https://foldingburritos.com/blog/kano-model>
- 4、《敏捷软件开发实践 估算与计划》Mike Cohn
- 5、<https://baike.baidu.com/item/KANO%20%E6%A8%A1%E5%9E%8B/19907824?fromtitle=%E5%8D%A1%E8%AF%BA%E6%A8%A1%E5%9E%8B>
- 6、<http://www.woshipm.com/pmd/3209305.html>

相对权重优先级排序

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

相对权重是一种通过评估特性的相对价值、成本和风险，并可为其分配权重来计算优先级的一种方法。

实践出处

相对权重方法由Karl E. Wiegers在1999年的一篇文章“First Things First: Prioritizing Requirements”中提出。这篇文章最初发表在《Software Development》1999年9月这一期。后来经过《Software Development》杂志的允许，重新刊登（有修改）在<https://www.processimpact.com/>上。相对权重方法借鉴了QFD的客户价值概念，即客户价值取决于存在特定产品功能时提供的客户利益和不存在该功能时支付的罚款。后来Mike Cohn在《敏捷估算与计划》一书中引用了这一方法，但Mike Cohn又对其进行了修改。本实践中是Karl Wiegers的原版。

为什么

相对权重优先级排序法将需求的价值从实现所带来的收益和不实现会招致的惩罚两方面进行结合考虑，并把成本和风险结合起来，同时可以配置各自的相对权重。既照顾到了与排序相关的最主要因素，又增加了非常大的灵活性。

何时使用

仅将此优先级方案应用于可协商的功能，即不在最高优先级类别中的功能。例如，您不会在该优先级分析中包含实现产品核心业务功能、关键产品差异化因素或遵守政府法规所需的功能，因为无论如何，都要包含这些功能。一旦您确定了产品发布必须包含的那些功能，就可以使用该模型为其余功能的相对优先级来做排序了。

如何使用

第1步：列出你希望在电子表格中优先考虑的所有需求、特性或用例。

第2步：用1-9估算每个功能为客户或业务提供的相对收益，1表示收益很小，9表示可能的最大收益。

第3步：估算如果不包括该功能，客户或企业将遭受的相对惩罚。同样，使用1-9，其中1表示基本上没有处罚，9表示非常严重的惩罚。

第4步：“总价值”列是相对收益和惩罚的总和。默认情况下，收益和惩罚的权重相等。你也可以更改这两个因素的权重。电子表格自动汇总特征值（总价值），并计算这些特征提供的产品总价值中归属于每个特征的百分比（价值%）。

第5步：估算实现每个功能的相对成本，还是用1-9。电子表格将自动计算每个功能的总成本百分比（成本%）。开发人员根据需求复杂性、所需用户界面工作的范围、重用现有设计或代码的潜在能力以及测试层级和所需的文档等因素来估算成本评级（相对成本）。

第6步：开发人员用1-9估算与每个功能相关的技术或其他风险的相对程度。估算值为1意味着你可以在睡梦中编程，而9表示严重担心可行性、是否有具备所需专业知识的员工，或者是否使用未经验证或不熟悉的工具和技术。电子表格将计算来自每个特性的总风险的百分比（风险%）。默认情况下，成本和风险的权重相等，每一项都与收益和惩罚条款具有相同的权重。您可以在电子表格中调整成本和风险权重。如果不想在模型中考虑风险，请将风险权重值设置为零。

第7步：在电子表格中输入估算值后，它会计算每个特性的优先级。

优先级列的公式为： $\text{优先级} = \text{价值\%} / (\text{成本\%} * \text{成本权重} + \text{风险\%} * \text{风险权重})$ 。

第8步：按计算的优先级按降序排列要素列表。列表顶部的特性在价值、成本和风险方面具有最有利的平衡，因此应该具有更高的实现优先级。关键客户和开发人员代表应审查完成的电子表格，以就评级和结果顺序达成一致。

相对权重	2	1	1	0.5					
特性	相对收益	相对惩罚	总价值	价值%	相对成本	成本%	相对风险	风险%	优先级
特性1	5	3	13	8.4	2	4.8	1	3.0	1.345
特性2	9	7	25	16.2	5	11.9	3	9.1	0.987
特性3	5	5	15	9.7	3	7.1	2	6.1	0.957
特性4	2	1	5	3.2	1	2.4	1	3.0	0.833
特性5	4	9	17	11.0	4	9.5	4	12.1	0.708
特性6	4	3	11	7.1	3	7.1	2	6.1	0.702
特性7	6	2	14	9.1	4	9.5	3	9.1	0.646
特性8	9	8	26	16.9	7	16.7	8	24.2	0.586
特性9	3	4	10	6.5	4	9.5	2	6.1	0.517
特性10	7	4	18	11.7	9	21.4	7	21.2	0.365
Totals	54	46	154	100	42	100	33	100	--

输出物

各特性优先级值

参考资料

- 《敏捷软件开发实践 估算与计划》Mike Cohn
- <https://www.processimpact.com/articles/prioritizing.pdf>



扫码查看详情

定义

加权最短作业优先 (WSJF) 是一种优先排序模型，用于对待开发的产品功能进行排序，它的计算方法是将工作的延迟成本除以工期。在最短的时间内能够提供最大价值（或延迟成本）的工作，被优先选择实施。

实践出处

WSJF最早于2009年，由Don Reinertsen在《Principles of Product Development Flow: Second Generation Lean Product Development》一书中提出。后来由Dean引入到规模化敏捷框架SAFe中进行了适当修改使用。

为什么

在基于流的系统中，不断对优先级做出调整可获得最大的经济回报。在这种情况下，要想获得最佳结果，靠的是工作排序，而不是理论上的单个工作的投资回报。通过WSJF，就可以驱动工作项随着时间的推移，动态排序到正确的位置，以获取最大价值。

何时使用

在做功能特性排序的时候使用。

如何使用

特性	用户-业务价值 +	时间紧迫性 +	风险降低 机会实现的价值 =	延迟成本 ÷	功能的规模 =	WSJF

- 每个参数的取值范围为1, 2, 3, 5, 8, 13, 20;
- 注意：每次做一行，从最小项开始，并给它赋值1;
- 每列必须至少有一个“1”;
- WSJF分值最高的特性优先级最高。

输出物

各特性WSJF值

参考资料

- 1、Reinertsen, Don. Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas Publishing, 2009.
- 2、<https://www.scaledagileframework.com/wsjf/by-sa/4.0/deed.zh>



扫码查看详情

定义

故事点是相对估算中的一种度量单位。设定某个基准故事的大小为1，其他故事的大小即为这个基准故事大小的倍数。通常建议使用斐波那契数列（1, 2, 3, 5, 8, 13, 20, ...）作为故事大小的取值范围。

实践出处

故事点最初是由Ron Jeffries作为极限编程（XP）的一部分提出来的，后来慢慢与其他敏捷框架（如Scrum）结合使用。

为什么

使用故事点估算：

- 估算速度更快，因为做相对比较比做绝对大小估算速度更快。
- 不同水平的人估算的结果更容易一致，因为相对比较受水平高低的影响非常小。
- 通常人为添加的水分相对小，相比之下，如果使用时间为单位进行估算会导致人为增加水分，因为在传统的管理机制下，通常会认为估算就是承诺，为了能够确保在承诺的时间内完成，估算者会填充很多水分。

何时使用

在需求梳理或者计划会议的时候进行估算时使用。

如何使用

- 1、选取一个所有人都清晰的、较小的用户故事作为基准故事，指定其故事点为1。
- 2、将其他故事的规模 and 这个基准故事进行比较，其他故事是基准故事的多少倍，那么故事点即为多少。
- 3、估算的时候要考虑如下因素：
 - 完成工作所需的预期工作量和专业知识；
 - 团队过去完成类似任务的经验；
 - 估算时的不确定性、疑虑和风险；
 - 完成任务所需的测试和错误修复；
 - 工作流程管理开销，包括沟通、会议和完成所需的管理工作。
- 4、通常结合使用估算扑克或者亲和估算来使用。

注：故事点是对故事整体进行估算，不应该对故事的不同部分分别进行估算之后再求和，这一点和以时间为单位的估算方式不同。

参考资料

<https://www.parabol.co/blog/what-are-story-points/>

亲和估算

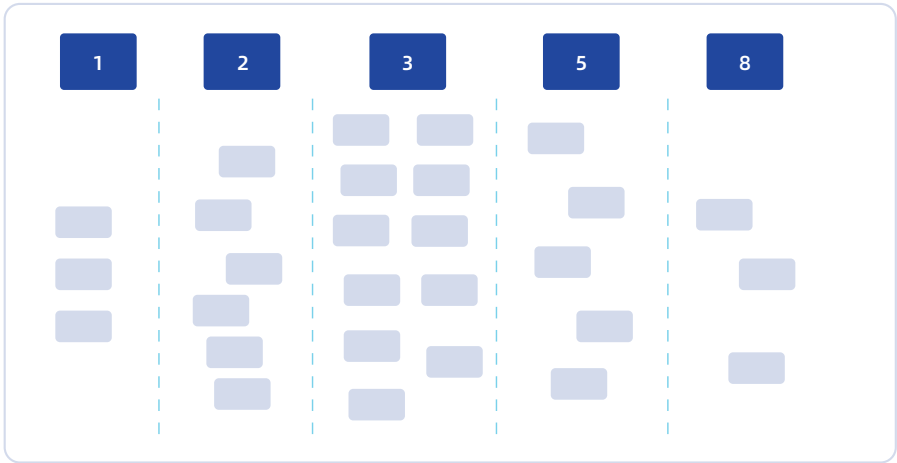
贡献人：徐东伟 露西 晏瑞宇



扫码查看详情

定义

亲和估算是将大量的条目按照规模相对大小依次顺序排列并分组，然后按组分配估算值的快速估算方法。



为什么

使用亲和估算，可以在短时间内对大量的需求条目进行估算，节省时间，提高效率。

使用亲和估算，也能够避免因为估算量大导致的估算基准偏离，让估算时间相距较远的条目的估算基准能够一致。

何时使用

当待估算的需求条目数比较多时（如超过25个），用亲和估算就比较高效。

如何使用

向团队成员宣读第一个条目，并放置在墙上。

阅读第二个条目，询问团队该条目比第一个条目大还是小，根据团队的反馈放到墙上对应的位置（越往右边越大，越往左边越小）。

读第三个条目，询问团队该条目与第一个、第二个条目的大小关系，然后放在墙上相应的位置。

然后将控制权移交给团队，由团队完成剩余项的亲和估算。

团队可以选择以同样的方式继续，即讨论一个，就往墙上放一个。然而，更快的方法是让每个团队成员根据自己的理解选择一个条目放到墙上他认为合适的位置。所有团队成员并行进行，直到所有条目都放到了墙上。

然后团队会为卡片检查分组情况，讨论大家认为放错组的条目，并在讨论完按需调整到正确的分组。

最后为各个不同的分组指定大小，通常建议使用斐波那契数列，也可以使用其他相对估算方法如T-Shirt尺寸。

输出物

各特性估算值

参考资料

- 1、<https://www.pmi.org/learning/library/agile-project-estimation-techniques-6110>
- 2、<https://www.thatsintelligence.com/blog/affinity-estimation>
- 3、<https://publicagile.org/agile-playbook/agile-planning/affinity-estimation/>

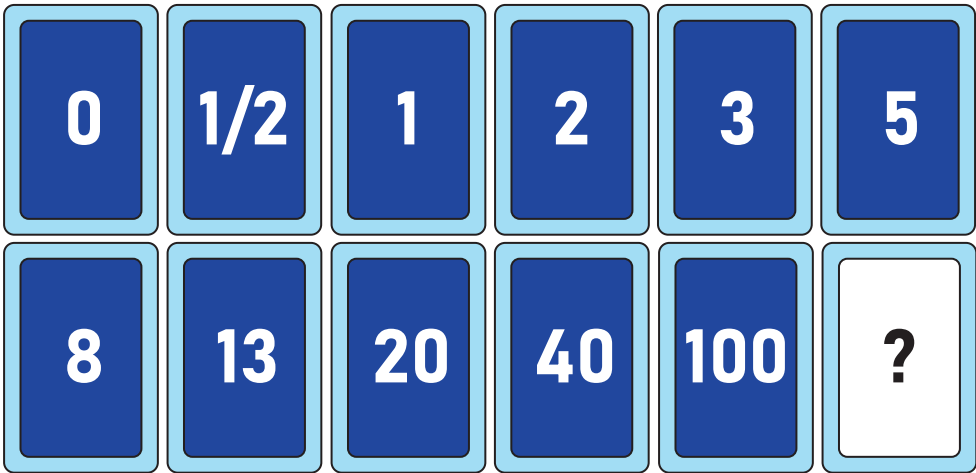
计划扑克

贡献人：徐东伟 李虎 晏瑞宇



定义

计划扑克是一种基于共识的游戏化估算技术。



小组成员选择带有自己估算数字的卡牌，面朝下。在所有人完成选择之后，大家同时展示卡片上的估算数字，然后讨论这些估算。

计划扑克是宽带德尔菲法的一种变体，同时亮牌的方式，可以避免自己的估算无意中受到他人的锚定。它最常用于敏捷软件开发，尤其是Scrum和极限编程。

实践出处

该方法最初由James Grenning于2002年定义并命名，后来由Mike Cohn在《敏捷估算与计划》一书中推广，Mike Cohn的公司为计划扑克这一术语和以计划扑克为名的在线工具注册了商标。

为什么

使用计划扑克，大家可以在不受其他人干扰/锚定的情况下做出自己的估算，大家的真实想法能够得到充分展示。

何时使用

通常在需求梳理活动中对需求/用户故事进行估算的时候使用计划扑克。

如何使用

会议由一位不参与估算的主持人主持，在估算过程中，每个估算者都会得到一副完全相同的卡片。步骤如下：

- 1、产品负责人简要概述一个待估算的需求/用户故事。团队有机会提问和讨论，以澄清假设和风险。
- 2、每个人正面朝下放置一张卡片，代表他们的估算。估算使用的单位可以是持续天数、理想天数或故事点。在讨论过程中，绝对不能提及与特性规模相关的数字，以避免锚定。
- 3、所有人同时翻牌展示自己的估算。
- 4、给高估算值和低估算值的人一个机会，阐述他们所选估算值的理由，然后讨论继续。
- 5、重复2-4，直到达成共识。

注：估算的目的不仅仅是为了得到一个关于需求规模的估算值，更重要的是，通过估算的过程，可以看到大家对于需求是否有一致的理解。如果不一致，就通过讨论和澄清达成一致，从而尽量减少因后期修复带来的成本增加。

输出物

各特性估算值

常见问题解答

为什么估算扑克上的数值不是标准的斐波那契数列？

标准的斐波那契数列是1、2、3、5、8、13、21、34、55……但是目前绝大多数团队在估算时使用的斐波那契数列是1、2、3、5、8、13、20、40、100……数列的前面6个数字是一样的，但是从第7个数字开始，就完全不一样了，这是为什么呢？

Mike Cohn曾经在他的文章中提到，早期的估算他都是根据真实的斐波那契数列进行的（1、2、3、5、8、13、21、34、55……）。

这个数列越往后数字越大，也就说明了估算越不准确，所以对于21、34、55这样的估算值，很多人会觉得很奇怪：既然已经估不准了，为什么非要使用21，而不将其四舍五入为20或25呢？

于是Mike Cohn就接受了这个建议，之后他在敏捷实践以及授课中就都开始使用20而不是21了，并且一直持续到了现在。

后来他又尝试使用了40和100这两个数代替了数列的其他值，之所以这么使用，也是因为任务的粒度越粗，估算就越不准确，也就越不需要纠结到底是80、90还是100了。

后来在Mike Cohn的影响下，现在大部分公司在敏捷估算中都开始使用修正后的斐波那契数列了：1、2、3、5、8、13、20、40、100、∞。

参考资料

- 1、https://en.wikipedia.org/wiki/Planning_poker
- 2、<https://www.mountangoatsoftware.com/blog/why-the-fibonacci-sequence-works-well-for-estimating-by-sa/4.0/deed.zh>

需求跟踪矩阵

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

需求跟踪矩阵是一种主要管理需求变更和验证需求是否得到了实现的有效工具，借助需求跟踪矩阵，可以跟踪每个需求的状态。

需求跟踪矩阵							
项目名称：							
成本中心：							
项目描述：							
标识	关联标识	需求描述	业务需求、机会、目的和目标	项目目标	WBS可交付成果	产品设计	产品开发
001	1.0						
	1.1						
	1.2						
	1.2.1						
002	2.0						
	2.1						
	2.1.1						

为什么

1、变更影响分析

如果需求正在变更，跟踪链接会帮助我们了解与之相关以及有依赖关系的工件。这样就可以很容易验证这些工件，如果需要还可以对它们进行调整。大大降低了忽略相关工件的概率。

2、覆盖率分析

确保不忽略任何需求。

3、项目状态分析

跟踪项目状态是可能的：分析跟踪数据可以查看需求的完成状态。没有链接或跟踪链不完整的需求（例如，有实现但没有测试的需求）表明需要进一步的工作。缺失的链接显示哪些具体工件缺失，需要实现。

何时使用

在研发的整个过程中

如何使用

纵向跟踪矩阵，包括如下的3种：

- 需求之间的派生关系：客户需求到产品需求；
- 实现与验证关系：需求到设计，需求到测试用例等；
- 需求的责任分配关系：需求由谁来实现。

横向跟踪矩阵：需求之间的接口关系
在SEI的调查中达成的基本共识是：纵向跟踪是必须的。

对于纵向跟踪矩阵：

必需的：

- 客户需求与产品需求的跟踪；
- 产品需求与测试用例的跟踪；
- 100%的接口需求需要建立客户需求-产品需求-设计-编码-测试用例的跟踪矩阵；
- 全局性需求要建立跟踪矩阵，包括：客户需求-产品需求-设计-编码-测试用例的跟踪矩阵；
- 核心需求要建立跟踪矩阵。

并非必需的：

- 性能需求可以不建立跟踪矩阵；
- 不影响系统架构的功能需求。

由于在需求跟踪矩阵中，需求可能有很多项，设计、测试用例、代码等都有多项，所以建立和维护需求跟踪矩阵的工作量还是比较大、比较烦琐。对于变化频繁的项目，更是如此。

在实践中，为了简化需求跟踪矩阵的建立与维护工作，有的企业仅仅通过需求与设计、代码、测试用例的编号来实现跟踪，如需求为：r1,r2等编号，而设计的编号为：r1-d1,r1-d2,.....测试用例的编号为：r1-t1,r1-t2等等。需要注意的是需求与它们之间是多对多的关系，仅通过编号是无法实现这种关系的。如果不借助专门的需求管理工具，一般只能通过Excel来维护需求跟踪矩阵，工作量就是比较大。

输出物

需求跟踪矩阵

参考资料

- https://en.wikipedia.org/wiki/Requirements_traceability
- <https://baike.baidu.com/item/%E9%9C%80%E6%B1%82%E8%B7%9F%E8%B8%AA%E7%9F%A9%E9%98%B5/6155948?fr=aladdin>
- <https://www.softwaretestinghelp.com/requirements-traceability-matrix>

变更控制委员会

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

在软件开发项目中，变更控制委员会（CCB）是一个委员会，由主题专家（SME，如软件工程师、测试专家等）和经理（如质量保证经理）组成，他们决定是否对项目实施拟议变更。

为什么

变更控制委员会的主要目标是确保客户接受该项目。每次的变动都要在批准的情况下进行，从而管理各方的期望。

何时使用

当出现与原商定基线不同的需求变更时，就要提请变更控制委员会批准。

如何使用

变更控制委员会将审查与客户商定的原始基线要求的任何拟议变更。如果委员会同意任何变更，则该变更将传达给项目团队和客户，且要求以变更为基准。变更控制委员会的权限可能因项目而异，但变更控制委员会达成的决定通常被视为最终决定并具有约束力。

影响CCB决策的因素包括项目的开发阶段、预算、进度和质量目标。

参考资料

https://en.wikipedia.org/wiki/Change_control_board

礼宾MVP

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

礼宾MVP是全程使用手动的方式引导用户解决问题，和用户一起探索问题/解决方案，通过亲自与用户进行交互，对用户和你所提供的价值有很大的了解。与绿野仙踪MVP不同，在绿野仙踪MVP中，用户看到的是产品，不知道后面是人工操作。而礼宾MVP全程都是人和人的互动。

为什么

礼宾MVP不仅可以通过跳过产品建设节省大量的时间，还可以让你直接与你的早期客户接触，深入了解他们的需求，探索“人们是否真的需要我提供的东西？”这个问题的答案。

何时使用

如果你自己对客户问题的理解不太自信，并且希望与客户进行更深入的互动，请使用礼宾MVP。

如何使用

- 1、确定要验证的假设；
- 2、找到在其上验证假设的最终用户；
- 3、提供人工服务；
- 4、在服务的同时，与用户交流并考察假设是否与用户的真实需要匹配，同时在用户需要的深度和广度方面进行探索；
- 5、根据获取到的新信息迅速调整，再次验证（即回到步骤3）。

当验证成功后，将整个过程进行自动化处理形成产品。

示例：

Food On The Table 是礼宾部MVP的完美例子。这个网站向用户发送来自不同商店的菜谱和交易，这样用户可以在考虑自己选择的同时省钱。创始人团队最初只为一位客户服务，这位客户得到了礼宾服务。该公司首席执行官曼努埃尔·罗索（Manuel Rosso）每周都会亲自拜访该客户，而不是让客户与数字应用程序交互。因此，对于早期客户，创始人和副总裁将审查商店中的销售内容，并根据客户的偏好选择食谱。产品将每周送到用户家中，征求反馈。一旦模型变得可行，就把该过程进行了自动化。

参考资料

- 1、<https://www.yarandin.com/en/concierge-mvp>
- 2、<https://www.shortform.com/blog/concierge-mvp/>

绿野仙踪MVP

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

绿野仙踪MVP依靠人工服务客户，但最终用户不知道是人躲在后面，它给客户一个假象，以为这就是最终的产品。

为什么

绿野仙踪的MVP是测试假设的最快和最有效的方法之一，它不需要实际构建任何东西，就可以测试拟议的解决方案是否会给客户创造价值。由于测试是手动进行、低成本的，因此可以快速修改MVP，测试大量的假设，找到最有效的解决方案。

何时使用

如果你对问题有很强的理解，并且希望缩小特定解决方案的范围，请使用绿野仙踪MVP。请记住，这些MVP应该关注单个解决方案方面，而不是一套选项。你的测试越专注，结果就越有价值。

如何使用

- 1、确定要测试的假设；
- 2、构造未来产品和最终用户之间的交互界面；
- 3、后台操作全部采用手工的方式进行；
- 4、分析测试的数据；
- 5、根据测试的数据调整假设并继续测试，直至达成满意的效果。

示例：

《绿野仙踪》（WizardofOz）MVP的一个经典例子是Zappos——亚马逊于2009年以12亿美元收购的一家著名在线鞋类零售商。Zappos的历史始于其创始人尼克·斯文穆恩（Nick Swinmurn），他决定检验人们愿意在网上购买鞋子而不必试穿的假设。当时，这一点还不确定，因此Nick没有遵循更传统的方法建立库存，然后尝试销售，而是采取了不同的方法。他去当地的商店，给鞋子拍照，然后在网上做广告。如果有一双鞋卖了，他会回到店里买下来，然后寄给顾客。客户不知道Nick没有一双鞋，也不知道所有订单都是手工处理的。这个想法被证明是成功的，业务很快获得了发展势头。结果，遵循绿野仙踪MVP原则，Nick从无到有地创建一个价值10亿美元的业务，而无需进行任何重大投资或开发任何复杂的解决方案。

参考资料

- 1、<https://www.yarandin.com/en/wizard-of-oz-mvp>
- 2、<https://mvpworkshop.co/blog/minimum-viable-product-mvp-type>
- 3、<https://toolkit.highlinebeta.io/wizard-of-oz-mvp>

着陆页MVP

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

1、着陆页是一个单一的页面，它：

- 描述你的产品或服务。
- 说明使用你的产品或服务的一些优势（你的“独特价值主张”）。
- 包含一个按钮，让感兴趣的访问者点击阅读更多内容、加入邮件列表、立即购买或其他一些行动。

2、着陆页的目标不是销售某些东西或产生潜在客户，而是通过对不存在的商品和服务做出广告宣传，从而获得有效的学习，而在之前，这些商品和服务只存在于企业家的脑海中。

为什么

着陆页中包含的信息，说明了用户使用你的产品/服务能获得什么样的优势（即你独特的价值主张）。这可以让你在产品非常早期的时候去验证你的价值主张是否能够与客户产生共鸣。

着陆页可以在几个小时内创建，如果能够将着陆页发布到合适的平台上，可以用非常短的时间完成整个构建-度量-学习精益循环！

何时使用

在制作其他类型的MVP之前（这是一种pre-MVP），在Web上提供产品概念，测试产品的价值主张，并查看产品想法是否在正确的轨道上。

如何使用

好的着陆页会包含以下几部分：

1、标题

最有用的标题应该是直接的，它能够描述产品是什么，以及产品是做什么的，或预期的好处是什么。

2、价值主张

说明用户使用你的产品/服务可以获得的好处和优势，这是潜在客户想要购买的主要原因。

可以按照要点列出来，还可以结合图片或者视频进行展示。

3、引发联系的机会

尝试向潜在客户销售，或者你至少应该让潜在客户展示他们的购买意图。这就需要引发客户的购买行为，为此可能要包含购买按钮、联系电话或在线表单。

4、提供报价

提供报价及优惠，它可以是限时优惠，并提供额外的奖励，以吸引客户立即购买。

5、A/B测试

考虑对不同版本的着陆页进行A/B测试，从而识别出MVP中哪些元素对潜在用户有吸引力，哪些没有。

如果着陆页的任何版本都没有在吸引潜在客户方面做得特别好，那么很遗憾，你的产品想法是不可行的。然后是时候考虑一种不同的方法来实现你的产品创意了。

示例：

Buffer的乔尔·加斯科因（Joel Gascoigne）使用一个着陆页来测试他的产品概念，该产品可以帮助用户在预先设定的最佳时间自动向社交媒体中发布信息。Joel获得了120份注册，并直接与其中50人进行了交谈。Joel甚至在推出此列表中的产品几天后就有一位付费客户。现在Buffer每月的经常性收入超过100万美元。

参考资料

- 1、<https://openclassrooms.com/en/courses/4544561-learn-about-lean-startup/4703206-discover-the-4-types-of-minimum-viable-product>
- 2、<https://draftss.com/how-to-design-a-good-mvp-landing-page/>
- 3、<https://www.yarandin.com/en/landing-page-mvp>
- 4、<https://landerapp.com/blog/how-to-use-a-landing-page-to-validate-your-startup-idea>

电子邮件MVP

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

这是一种使用电子邮件对目标市场进行测试的方法，它相对更便宜、更快速。

为什么

电子邮件MVP成本低、快速且直接。

何时使用

如果你有一个现有的潜在客户网络，那么使用电子邮件MVP可能是比较适合的类型。如果你有独特的有价值的信息，你可以在电子邮件简报中分享，看看你会得到什么样的需求。

如何使用

- 1、确定要测试的假设；
- 2、利用合适的邮件服务平台向潜在的客户发送与要测试的假设相关的邮件，或者提供邮件订阅服务；
- 3、观察潜在客户的反应并调整邮件的内容。

当潜在客户的反应达到预期，就可以向产品投入更多精力打造细节。

示例：

Product Hunt：Product Hunt 是一个网站，上面有一个顶级科技产品和初创公司名单。但在其作为网站流行之前，Product Hunt最初只是一份简单的电子邮件简报。

随着订阅者数量的增长，一个由企业家、创始人和风险资本家组成的蓬勃发展的社区产生了对更全面网站的需求。Product Hunt能够使用他们的电子邮件简报来测试需求，并为产品信息资源建立一个社区，然后再将资金和资源用于构建一个成熟的网站。

AngelList：AngelList是一个面向投资者、企业家和那些希望为初创公司工作的人的网络，他们都希望围绕自己的想法和概念建立关系。就像Product Hunt一样，这个主要网站最初是以电子邮件列表的形式出现的，在这里，天使主义创始人向投资者介绍创业者。

一旦他们看到人们的兴趣在增加，他们就投入更多的资金和时间来建立一个更大的网络。如今，AngelList作为一个拥有350多万用户的网站，是初创公司创始人最受欢迎的资源之一。

参考资料

- 1、<https://www.linkedin.com/pulse/types-minimum-viable-products-series-email-campaign-michael-flowers/>
- 2、<https://steelkiwi.com/blog/what-mvp-and-why-it-necessary/>

碎片拼接MVP

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

碎片拼接MVP指的是推出新的产品或服务时，用第三方服务或应用程序来填补特定的空白，而不是花时间自己开发产品的那个方面，达到的效果是整体方案比其各个部分的总和更有价值。

为什么

- 使用这些第三方服务可以无需提前进行大量专有开发，即可对市场做出测试，成本更低、无需进行大量人员配置，并缩短上市时间。
- 由于可用的第三方服务几乎是无限的，有无穷无尽的可能性为你提供独特的工具或服务。
- 亲密使用高质量的产品可以为你开发自己的变体服务提供大量的见解。

何时使用

想要验证自己的想法，同时这个想法中的很多部分是可以通过第三方的服务协助进行的，就可以尝试使用碎片拼接MVP。

如何使用

- 首先，确定一个你想提供的新的或更好的服务或产品。
- 结合第三方服务，如聊天程序、文件和工作协作网站、邮件列表创建服务、照片分享页面，以及其他你可以轻松利用的服务，创造某种新的可行的产品体验。
- 类似于礼宾MVP，碎片拼接MVP也不依靠自动化来提供服务，你将成为各个碎片服务之间的连接，以创造一个“完整”的产品。
- 收集对碎片拼接产品或服务的第一手的客户反馈和需求，以决定是否要继续前进。
- 在未来，尽管有可能重新创建你在MVP中所依赖的服务，但也可能有必要与这些第三方服务合作。

参考资料

<https://www.linkedin.com/pulse/types-minimum-viable-products-series-piecemeal-michael-flowers>

众筹MVP

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

通过众多人的小额投资来资助你的商业理念。

为什么

使用众筹MVP有如下好处：

- 它相对便宜
没有现有客户或客户数据的公司可以使用众筹平台以非常经济的方式测试全新概念。众筹活动是与早期采用者签约并从感兴趣的受众那里获得直接反馈的好方法。它有助于创建需求。基于“奖励”的众筹可以让买家和卖家互通有无，即买家更愿意去了解和尝试新产品，而卖家可以更好了解买家。早期采用者同意通过购买“奖励”来资助创意，这些奖励通常是产品和服务的初始版本，其价格低于预期的未来零售价格。现在有很多创始人通过众筹成功建立了需求，而之前没有受众。
- 它可能会形成一个社区
除了资金之外，众筹平台还有助于为早期风险投资收集可能最有价值的战略资产：一个支持者社区。这些人远不仅仅是消费者（普通的消费者可能会积极使用产品，但在产品开发过程中仍然保持被动），支持者是真正的业余爱好者，他们有主人翁意识，通常愿意为项目的成功贡献更多的资金。他们可能会向企业家提供建议，让他们了解其市场的新兴需求，引入有价值的联系，并将现有和未来的产品介绍给他们的朋友和网络。
- 它可能是筹集更多资金的起点
一旦一个项目得到资助，募捐者在随后的众筹活动中更容易吸引支持者。

何时使用

当要开发一款产品，但缺乏资金的情况下，可以使用这种方法来筹集资金。要注意，此时这个产品的概念是要经过慎重调研且有强有力的故事和推进计划的。

如何使用

- 1、提出你想创建的概念或原型；
- 2、使用Kickstarter或Indiegogo等众筹服务；
- 3、那些有兴趣的人将捐钱，以便使产品成为现实；
- 4、这些钱将用于开发和向市场推出产品。

参考资料

- 1、<https://medium.com/@mdotflow22/types-of-minimum-viable-products-series-preorder-crowdfunding-c7b609379366>
- 2、<https://readyfundgo.com/crowdfunding-mvp/>

单一特性MVP

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

单一特性MVP是只包含一个单一特性的MVP，它是一个软件产品，旨在测试用户对一个特定特性的接受程度。这是最传统的MVP方法。

为什么

单一特性MVP实现了产品最核心的功能，体现了产品的核心价值主张，使用它对市场进行测试，有助于决策未来产品方向的调整。

何时使用

如果你确定了未来产品的核心功能，就在一个单一特性的MVP中实现它，以检查它在实际市场上的可行性。

如何使用

- 1、确定产品的核心价值主张。
- 2、将符合核心价值主张的核心功能实现出来并发布。
- 3、收集市场的反馈。
- 4、根据市场的反馈进行调整。

参考资料

- 1、<https://keyua.org/blog/types-of-minimum-viable-product/>
- 2、<https://www.linkedin.com/pulse/types-minimum-viable-products-series-single-feature-michael-flowers/>
- 3、<https://www.softwareworld.co/why-and-how-you-should-develop-a-single-feature-mvp/>

A/B测试

贡献人：徐东伟 晏瑞宇



扫码查看详情

定义

当需要在两种方案之间做出选择或寻找问题的根本原因时可以使用A/B测试。

实践出处

生物学家和统计学家罗纳德-费希尔（Ronald Fisher）在20世纪20年代进行了随机对照实验。他研究出了基本的数学运算和原则，并把这个想法变成了一门科学。

费舍尔进行了几个农业试验，以寻找一些基本问题的答案，比如：如果换了肥料或使用更多的肥料会怎样。他所介绍的原则被证明是有效的，科学家们在20世纪50年代初正式开始在医学领域进行临床试验。营销人员在20世纪60年代末采纳了这种技术。他们想评估一些直销活动，即：使用个性化的信件或明信片是否能够帮助提升销售量。

然而，A/B测试的形式以前和现在有所不同。它在20世纪90年代中期形成了目前的形式。虽然现在使用相同的概念，但已经转移到虚拟环境下，并且是实时进行的。

何时使用

想要在两种方案之间做出选择时，或者寻找问题的根本原因时都可以用A/B测试。

如何使用

- 1、确定要解决的问题。确保你确定一个要解决的具体问题。
- 2、分析用户数据，用来识别问题可能出在哪里。
- 3、制定要测试的假设。决定你到底要测试什么，以及你要如何测试。将你的未知因素缩小到1或2个，至少开始是这样。然后，你可以确定改变哪个或哪些元素可能会解决你所面临的问题。
- 4、对假设进行测试。开发一个新版本，实现你的想法。然后在该新版本和你的目标受众的当前页面之间进行A/B测试。
- 5、分析数据。一旦测试结束，看一下结果，看看你的项目的新版本是否导致了任何明显的变化。如果没有，试着测试一个新的元素，直至得到期望的结果。

输出物

测试报告

参考资料

- 1、https://en.wikipedia.org/wiki/A/B_testing
- 2、<https://mailchimp.com/marketing-glossary/ab-tests/>
- 3、<https://adoric.com/blog/a-b-testing-how-it-works/>

开发

编码

代码评审

分支策略

持续集成

编码规范

贡献人：张鹏 李虎 晏瑞宇 徐东伟



扫码查看详情

定义

编码规范是针对特定编程语言制定的一组指导原则，它为使用该语言编写程序的每个方面推荐编码风格、实践和方法。

为什么

1、降低软件维护成本

在软件的整个生命周期中通常需要投入高额的维护成本。编码规范使不同工程师编写的代码具有统一的风格，提高了代码的可读性和可维护性，因此工程师们可以更快更彻底地理解代码。

2、提高软件质量

编码规范中明确地描述了在编码风格、原则、实践、方法以及经验等方面的要求和建议，切实地为如何提高软件质量指引了方向。

3、提高软件开发效率

编码规范作为代码集体所有的重要基础，可以帮助团队打破技术壁垒，促进知识共享。团队积极合作关注整体，有助于代码的重用，冗余的消除，在降低复杂度的同时减少浪费，提高开发效率。

何时使用

McConnell描述了编码规范实践的理想状态，在开始编程之前建立编程规范。否则以后几乎不可能修改代码来匹配它们。在已有编码规范的前提下，仍要保持对编码规范的调整和优化。针对尚未制定编码规范的情况，推荐尽早制定。

如何使用

可以结合以下几个方面参考业界的编码规范，然后结合自身实际情况进行裁剪或补充：

1、排版

- 符号的使用规则，如括号，制表符
- 缩进的使用规则
- 空行，代码逻辑的分段

2、命名

- 命名的基础规则，例如符合当前业务场景，命名易懂等
- 类、函数、变量等不同实体的名称的格式，例如驼峰等
- 名称的最大长度

3、注释

- 使用时机
- 格式
- 对注释的正确性、及时性的要求

4、编程原则举例

- 抽象原则
- 单一职责
- 里氏替换原则
- 开闭原则
- KISS原则

5、编程实践举例

- 代码评审
- 代码走查
- 重构
- 结对编程
- 测试驱动开发
- 持续集成

6、架构最佳实践

- 架构的选型
- 文件的组织
- 架构使用规范

7、常见问题举例

- 页面样式应写在css文件中
- 避免使用inline styles

编码规范内容确定后，应尽量将规则的目的与意义描述清晰，并且应对编码规范进行版本控制管理。编码规范作为编程的指导准则应受到尊重，每个工程师都应该严格遵守。

与特定语言相关的编码规范，可以参考Google Style Guide，该编码规范包含的语言有：

C++	Java	HTML/CSS	Common Lisp
C#	Python	JavaScript	Vimscript
Swift	R	TypeScript	
Objective-C	Shell	AngularJS	

大家可以根据自己团队的语言，选择合适的编码规范。

输出物

编码规范文档

参考资料

- 1、https://en.wikipedia.org/wiki/Coding_conventions#Common_conventions
- 2、https://en.wikipedia.org/wiki/Programming_style
- 3、[https://en.wikipedia.org/wiki/Comment_\(computer_programming\)](https://en.wikipedia.org/wiki/Comment_(computer_programming))
- 4、https://en.wikipedia.org/wiki/Indentation_style
- 5、https://en.wikipedia.org/wiki/Characters_per_line
- 6、[https://en.wikipedia.org/wiki/Naming_convention_\(programming\)](https://en.wikipedia.org/wiki/Naming_convention_(programming))
- 7、https://en.wikipedia.org/wiki/Coding_best_practices
- 8、https://en.wikipedia.org/wiki/Category:Programming_principles
- 9、<https://github.com/google/styleguide>

结对编程

贡献人：栾绍晨 徐东伟 晏瑞宇



扫码查看详情

定义

结对编程是一个敏捷软件开发的技巧，由两名程序员在一个计算机前共同工作。一个作为“驾驶员”负责编写代码，而另一个作为“观察员”或者“领航员”，负责审查“驾驶员”写的每一行代码。这两个程序员会经常互换角色。

实践出处

《人月神话》的作者Fred Brooks声称他在20世纪50年代的时候与同样是研究生Bill Wright一同编程过。他说：“我们一起创造了1500行没有缺陷的代码。它第一次就可以正确运行。”

到了20世纪80年代早期，Larry Constantine参观了P.J. Plaugher的软件公司。Larry发现了一个房间里每一个电脑前都有两个程序员在工作。他把这个称作“动态二重奏”。而这些程序员产出的代码是没有缺陷的。他说：“两个程序员串联在一起并不是累赘，而是一个最直接的提升效率和质量的方式。”

后来在1998年，在克莱斯勒实践的极限编程中，结对编程是它的核心实践之一。

为什么

结对编程可以在不影响交付事件的前提下提高软件质量。这是反直觉的，但是两个人在一台计算机上工作与两个人单独工作将产出一样多的功能，并且前者的质量还会更高。伴随着质量的提高，可以给项目后期提供更多的节省。

通常情况下人们倾向于关注总体思路而不是具体细节。而结对可以让程序员更快的发现错误。没有任何两个程序员会以相同的方式看待代码。这通过分配特定角色——“驾驶员”和“领航员”来辅助。

将两个角色拆分可以给代码带来两种不同的视角。“驾驶员”的想法应该更具有战术性，聚焦于细节，聚焦于当前正在写的代码；而“领航员”会思考更多具有战略层面的内容，他们会在脑海中绘制更大的蓝图。

在结对编程中，开发人员需要讨论代码并提出问题。提出正确的问题可以挑战假设并引发新的观点。因此结对编程不仅仅是代码，也是积极主动的学习。表达自己的想法有助于帮助我们达到正确的理解和共识。

何时使用

- 遇到复杂问题时

结对编程可以提高开发人员在处理概念上复杂的任务时的工作表现，因为这些任务的细节可能会非常多，以至于单个开发人员的处理速度会显著下降并且犯错误的可能性会更高。而有另一个开发人员在旁边会减少这些问题的发生。

- 当需要加速重要的任务时（例如：为了达成冲刺目标）

为了尽快达成冲刺目标，当冲刺待办事项列表的任务已经为空，但是还有和冲刺目标相关的任务正在进行的时候，程序员需要结对一同处理这些任务，而不是去产品待办事项列表中去领取新的任务进来。这可以让两个开发人员的全部开发能力能够承担冲刺中最关键的工作。

- 需要培训时

结对编程对于新手上手工作和流程非常有效。一个有经验的开发人员可以和一个新人结对来快速地、有效地分享项目上的

信息和常用的研发技巧。结对还有助于新人保持参与度并确保他们自己编写代码，而不是将工作留给有经验的开发人员。

- 遇到瓶颈时

当可用的开发人员多于任务数量时（例如，任务有依赖关系或者有阻碍的任务时），配对可以让多个开发人员的全部力量集中在需要处理的任務上。

还有其他的场景欢迎你的补充...

如何使用

使用结对编程的最佳方式就是并排坐在显示器前，经常互换键盘和鼠标。两位程序员都专注于正在编写的代码。

而“领航员”是观察的角色。当“驾驶员”在敲代码的时候，“领航员”会随时随地的审查代码、提出建议并且分享他的想法。同时，“领航员”应该关注更大的问题、缺陷漏洞并且记录下潜在的下一步或者障碍。

在审查的过程中，“观察员”同时需要考虑工作的战略性方向，提出改进的意见以及应对将来可能要解决的问题。这样做为了让“驾驶者”将所有的注意力集中在完成当前任务的“战术”方面。

一个常见的流程如下：

- 团队可以从一个定义合理且清晰的任务开始。
- 一次设立一个小目标并达成一致。这可以通过单元测试，提交备注(commit message)或者写到便签纸上来进行定义。
- 经常性的通过切换键盘使用权和角色。这份共享的积极参与可以让能量提升，并且让大家更好地学习和理解事物。
- 作为“领航员”，需要避免使用“战术性”思维，也就是不要关注于细节。将细节留给“驾驶员”。“领航员”的工作是退后一步，给你的同伴补充更多的“战术性”思维。把下一步要做的内容、潜在的障碍和写在便签纸上的想法先放一放，等小目标完成了之后再沟通。这么做是为了不中断“驾驶者”的思路。

实战案例

结对修改线上问题

2012年左右麦哲思科技在为某客户咨询时实践了该做法。

问题：系统上线后，紧急修改时往往修改了一个Bug，又引入了新的Bug，导致进一步的生产事故。

解决方案：系统上线后，遇到了修改任务，无论是需求变更还是缺陷修复，此时要安排两位开发人员守着一台电脑，互相协商着修改代码，两人达成一致后，才可以动手修改代码。

参考资料

- 1、https://en.wikipedia.org/wiki/Pair_programming
- 2、<http://www.extremeprogramming.org/rules/pair.html>
- 3、<https://martinfowler.com/articles/on-pair-programming.html#Styles>
- 4、<https://hinchman-amanda.medium.com/debunking-the-myths-of-pair-programming-7db756326b4c>

定义

测试驱动开发，英文全称Test-Driven Development，简称TDD，有广义和狭义之分，常说的是狭义的TDD，也就是 UTDD（Unit Test Driven Development）。广义的 TDD 是 ATDD（Acceptance Test Driven Development），包括BDD（Behavior Driven Development）和Consumer-Driven Contracts Development等。本文所说的TDD是指狭义上的TDD，也就是“单元测试驱动开发”。TDD是极限编程中的一项核心实践和技术，也是一种设计方法论。它要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。这有助于编写简洁可用和高质量的代码，并加速开发过程。

实践出处

TDD由极限编程中倡导，以其倡导先写测试程序，然后编码实现其功能得名，始于20世纪90年代，一般认为是Kent Beck在2000年左右提出的，但Kent Beck自己也说他并不是创始者，他只是把一个人已经在用而他本人觉得很好的开发方法系统地整理出来，让大家更容易理解、使用而已。

为什么

TDD要求在正式开始代码开发工作之前首先进行测试用例以及对应自动化测试脚本的编写，通过自动化脚本的运行，可以对软件的更改更加有信心。此外，TDD可以让不熟悉代码细节的程序员灵活、自信地改动源代码。TDD具有以下优势：

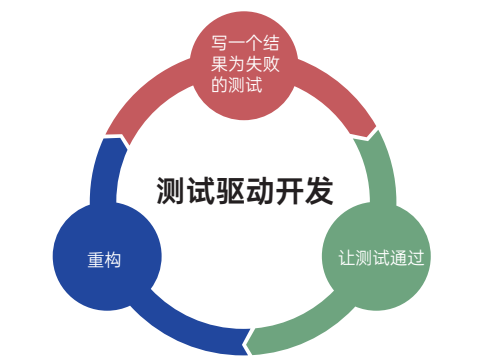
- 1、减少Bug；
- 2、开发出高可用软件；
- 3、开发出可扩展的软件；
- 4、减少冗余代码、简化代码；
- 5、帮助监测代码；
- 6、可以作为代码文档；
- 7、可快速得到反馈，缩短从开发到发布的周期。

何时使用

TDD可用于新项目的研发以及对已有项目进行优化、重构过程中

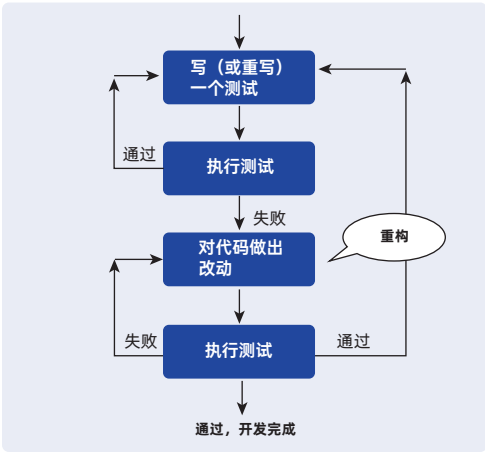
如何使用

TDD 的基本流程是：红，绿，重构。



详细来说有以下几步：

- 1、编写测试脚本：程序员根据用户故事以及测试用例编写满足当前需求以及规范的单元测试脚本，此处应注意，所编写的测试脚本应尽可能地小。
- 2、执行测试脚本：运行测试脚本，以确认它会失败。是的，一开始它会失败，如果一开始没有失败，这意味着肯定有什么地方已经错了，你可能正在复制已经存在的代码，或者在不知不觉中使用与程序中其他地方相同的变量名。
- 3、编写最少的代码来通过测试：只要能够通过测试，无论所编写的代码是否优雅亦或是硬代码都是可以接受的，所有的代码都将在第五步中得到优化。注意：不可以有超出测试范围的代码，并且只有当前用例的测试脚本完全通过之后才能对下一个用例进行测试脚本的编写。
- 4、所有程序代码均可通过测试脚本的测试：如果有任何的失败，那么必须对相关代码进行修正，直至最终通过测试。这一点能过确保相关代码符合测试需求，同时不会影响任何现有的功能逻辑。
- 5、按需进行重构：每次重构后都要进行测试，以确保之前的功能都是完善的，没有受到影响。
- 6、重复上述5个步骤：每个新功能点都需要重复上述五个操作步骤，需要注意测试单元应该非常地小，测试内容是可以追加的，而且，应该经常性地代码的提交。在这种条件下，如果有任何的测试失败，程序员可以很轻易的进行代码的撤销或者恢复，而不是花费大量时间进行代码调试。



延伸阅读

<https://www.youtube.com/watch?v=CHTep2zQVAc>

参考资料

- 1、Test Driven Development (TDD) by KentBeck
- 2、https://en.wikipedia.org/wiki/Test-driven_development

行为驱动开发

贡献人：王连杰



扫码查看详情

定义

行为驱动开发（Behavior-driven development，缩写BDD），是敏捷领域中测试驱动开发（TDD）的一种延伸技术，它旨在鼓励软件项目中涉及到的人员角色（包括开发者、QA和非技术人员或商业参与者等等）之间的协作。

BDD帮助敏捷研发团队把精力集中在识别、理解和构建跟业务目标有关的产品特性上面，并让敏捷研发团队能够确保识别出的产品特性能够被正确设计和实现出来。

引入BDD的软件研发团队通过充分的交流沟通和待实现的产品功能的使用场景举例，来帮助研发团队理解产品特性对业务的价值。BDD采用更容易测试的软件需求描述方式鼓励需求分析人员、软件开发人员、测试人员密切协同开展软件产品研发工作。同时BDD工具可以帮助把用BDD风格描述的业务需求转换成自动化测试脚本，让软件开发人员同步验证自己编写的代码是否满足业务需求描述的产品特性，并在验证软件产品特性的同时形成软件产品特性文档。从而实现了产品研发文档与软件产品代码编写的同步更新。

实践出处

2003年11月，由Dan North命名于伦敦，在其发表的“敏捷规格，BDD和极限测试交流”一文中，Dan North对BDD给出了如下定义：BDD是第二代的、由外及内的、基于拉动(pull)的、多方利益相关者的(stakeholder)、多种可扩展的、高度自动化的敏捷方法。它描述了一个交互循环，可以具有带有良好定义的输出（即工作中交付的结果）：已测试过的软件。同年，Dan North创建了第一个BDD框架JBehave[https://jbehave.org/]

2007年，Dan North为Ruby创建了一个名为RBehave的故事级BDD框架，该框架后来被集成到RSpec项目中。

2008年，参与了围绕BDD进行的首轮讨论的克里斯马茨，提出了特性注入(Feature Injection)的想法，BDD可以覆盖分析空间并提供从初期的展望到编码和发布的整个软件生命周期过程的改造。

2010年，David Chelimsky、Dave Astels、Bryan Helmkamp、Dan North，Zach Dennis五个人，以合作开发的RSpec为基础，一起撰写出版了“The RSpec Book: Behavior Driven Development with RSpec, Cucumber, and Friends”这本书。

2013年在对Dan North 的一次采访中BDD又被解释为：它使用例子来说明应用程序的行为..... 并针对这些例子进行交流。

RSpec中的第一个基于故事的框架后来被Aslak Hellesøy于2008年开发的Cucumber所取代。

Capbara是目前比较流行的BDD测试框架之一，Capbara通过模拟真实用户如何与您的应用程序交互来帮助您测试Web应用程序。它与运行测试的驱动程序无关，并带有内置的Rack::Test和Selenium支持，并能通过外部gem支持WebKit。

为什么

BDD诞生的初衷是去重新思考单元测试和验收测试的方法，以期避免曾经出现过的问题。通过与干系人的讨论取得对预期的软件行为的清醒认识。通过用自然语言书写非只有程序员可读的测试用例，扩展了测试驱动开发（TDD）方法。行为驱动开发（BDD）人员使用统一的领域语言来描述他们的代码的目的。这让开发者得以把精力集中在代码应该怎么写，而不是技术细节上，而且也最大程度的减少了将代码编写者的技术语言与商业客户、用户、干系人、项目管理者等的领域语言之间来回转换、翻译的代价。

BDD VS TDD

TDD发现的问题

- 由于代码重构时，必须相应去重构测试过程，导致测试资产极其脆弱，同时自动化测试资产的维护成本很高；
- 开发人员与团队其他成员之间的沟通问题——一些开发人员发现很难有效地描述他们每天编写的自动化测试这种缺乏沟通可能导致其他团队成员质疑和争论此类活动的价值；
- 开发人员很难估计他们何时会完成所有的自动化测试——因为答案是“取决于”，这取决于编写的类和方法的数量；
- 测试人员的重复测试工作——由于开发人员实现的自动化测试通常对团队中的非开发人员不可见，因此测试人员通常执行许多可能已经被开发人员的自动化测试覆盖的测试。

采用BDD的优势

- 团队事先就要实施的行为达成一致，之后就没有讨论任何测试价值的余地；
- 团队预先决定哪些行为由开发人员实现，哪些行为由测试人员执行；
- 团队可以更准确地向管理层提供估计——因为测试不依赖于编码的类和方法的数量；
- BDD通常有较少的测试资产并且这些资产相对比较健壮。所以当对代码进行重构时，要重构的测试代码就没有那么多。另外虽然BDD的测试资产较少，但仍然可以类似于TDD那样保证测试覆盖率 - 因为BDD的测试对象是整体行为 - 而不是单个方法或功能；
- 在BDD中每一个软件组件的测试过程都是透明的，包括测试人员和开发人员在内的所有人可以看到，因此避免了TDD那样的重复性测试工作。
- BDD通常有较少的测试资产并且这些资产相对比较健壮。所以当对代码进行重构时，要重构的测试代码就没有那么多。另外虽然BDD的测试资产较少，但仍然可以类似于TDD那样保证测试覆盖率：因为BDD的测试对象是整体行为，而不是单个方法或功能；
- 在BDD中每一个软件组件的测试过程都是透明的，包括测试人员和开发人员在内的所有人可以看到，因此避免了TDD那样的重复性测试工作。

如何使用

BDD的做法包括：

- 确立不同干系人要实现的远景目标；
- 使用特性注入方法绘制出达到这些目标所需要的特性；
- 通过由外及内的软件开发方法，把涉及到的干系人融入到实现的过程中；
- 使用例子来描述应用程序的行为或代码的每个单元；
- 通过自动运行这些例子，提供快速反馈，进行回归测试；
- 使用“应当(should)”来描述软件的行为，以帮助阐明代码的职责，以及回答对该软件的功能性的质疑；
- 使用“确保(ensure)”来描述软件的职责，以把代码本身的效用与其他单元(element)代码带来的边际效用中区分出来；
- 使用mock作为还未编写的相关代码模块的替身。

行为规范的用户故事

开发人员与干系人之间的交互要写成双方都能懂的用户故事，用户故事的一般构成是：

- 标题：一个显式的标题。
- 叙述：一个简短的介绍性的部分使用以下结构：
 - 作为一个：人或角色将受益于功能；
 - 我想要的：功能；
 - 这：利益或价值的特性。
- 验收标准：每个特定的描述场景下面的叙述结构：
 - 鉴于：初始上下文的场景中,在一个或多个条款；
 - 当：事件触发场景；
 - 然后：预期结果，在一个或多个从句。BDD没有任何正式的要求去指导如何去写用户故事，但它坚持认为每个团队使用BDD时，需要想出一个简单、标准化的格式去记录包括上面元素的用户故事。

一个非常简短的例子格式：

标题：退货和换货进入库存。

作为店主，我想在退货或换货时将商品添加回库存，以便跟踪库存。

场景1：退回退款物品应添加到库存中。考虑到一位客户之前从我那里买了一件黑色毛衣，而我的库存中有三件黑色毛衣，当他们退回黑色毛衣要求退款时，那么我的库存中应该有四件黑色毛衣。

场景2：交换的物品应返回到库存。考虑到一位客户之前从我这里购买了一件蓝色衣服，我库存中有两件蓝色衣服和三件黑色衣服，当他们将蓝色衣服换成黑色衣服时，我库存中应该有三件蓝色衣服和两件黑色衣服。

输出物

测试用例、测试结果

参考资料

- 1、https://en.wikipedia.org/wiki/Behavior-driven_development
- 2、<https://lizkeogh.com/2007/06/13/bdd-tdd-done-well>
- 3、<https://www.methodsandtools.com/archive/enterprisebdd.php>
- 4、<https://blog.csdn.net/winteroak/article/details/81585299>

集体代码所有权



扫码查看详情

贡献人：王岩 薛健 徐东伟 鲁飞

定义

集体代码所有权是一种明确的约定，即“每个”团队成员不仅被允许，而且实际上有责任，根据需要对“任何”代码文件进行更改：可以是完成开发任务、修复缺陷，也可以是改进代码的整体结构。集体代码所有权制下，每个人都对代码的质量负有责任。

实践出处

集体代码所有权是现代敏捷软件组织的基本实践。它在20世纪90年代后期通过极限编程 (XP) 引入大众。

为什么

集体代码所有权的好处：

1、知识共享

当只有一个人负责某些模块或功能时，就会形成知识孤岛。这导致关键知识仅限于一小群人，或者更糟糕的是，仅限于某个人。

当团队集体负责代码时，每个人都会学习。随着越来越多的人拥有能够有所作为的知识，产品质量必然会提高。

2、代码质量和更好的编码风格

有很多人为代码做出贡献时，这会使得代码的整体质量更好。当然，这是假设他们关心代码改进。在一个模块中只有一个人编码，他对自己的工作不会有任何建设性的批评。我们不能忽视编码风格——每个人都有不同的风格，有些人的编码比其他人的更容易理解，有些人的编码比其他人的更好、更高效。当整个团队一起为代码做出贡献时，它会在不依赖个人编码风格的情况下发展，因此更容易被更多的人理解。

3、不依赖单个人

举个幽默的例子，如果负责某部分代码的人中了彩票并离开公司怎么办？团队会挣扎求生吗？如果答案是肯定的，那么这就是开始给团队集体赋予责任的原因之一。这有时被称为“巴士效应”或“彩票效应”。

4、高效有用的代码审查

当只有一个人知道某个功能或模块中的代码时，代码审查就变成了一场闹剧。除了能够在高层次上进行抽象评论之外，无法提出真正的改进建议。另一方面，当代码评审中的每个人都对代码更加熟悉时，每个人对代码评审做出贡献都是非常有益的，讨论时会有不同的想法和改进出现，并且可以产生最佳的解决方案。

5、学习范围广

当公司在开发团队中创建专门的角色和职责时，会导致他们一直做重复的工作，大多数人的技能发展就会停滞不前。持续的学习才能使开发人员的技能保持优秀，思维持续活跃。通过与更好的开发人员进行持续的交互和讨论，他们可以快速的学到很多东西。

代码集体所有权会带来上述积极影响，但仍有以下方面值得考虑：

1、谁负责

当每个人都有责任的时候，就没有人真正负责了，在集体所有权里面，我们无法找到单独的个体为代码负责。

2、责任和动机

如果说成功人士与平庸人士的区别只有一件事，那就是成功人士总是希望承担更多责任。他们从来都不是只做自己的工作就回家的人。他们想进一步改进，他们的热情让他们更加努力。如果更熟练的开发人员没有被赋予更多的责任或挑战，这将使他们失去动力。将民主与个人才华分开是一把双刃剑。

3、常见误用

当考虑集体代码所有权时，集体就是团队。明确地说，“团队”是每天在共享代码库中依赖于协作开发技术进行协作更改的小组。通常，多团队组织将集体代码所有权误解为任何开发团队中的任何人都可以在任何时间对代码库的任何部分进行任何代码更改。这更像组织范围内的代码所有权，它通常会对软件的质量和团队的士气造成灾难性的影响。

4、组织范围的代码所有权问题

随着新功能的创建，代码需要开发和更改来支持这些新功能，因此长期关注代码库是非常重要的。拥有一个专注于代码库健康和远景的团队，可以实现迭代和紧急技术，比如进化设计和机会重构。允许任何人在没有代码远景或代码环境的情况下对代码库进行任何更改，通常会导致软件组织的混乱，并导致缺陷转移到生产中。

公司依赖于组织范围内的代码所有权有几个原因：最常见的罪魁祸首是系统的设计和体系结构。通常，组织在支撑没有功能点分离的单一应用程序时采用组织内代码所有权。这导致了一个大问题，当代码没有边界时，就不存在能够支撑集体代码所有权的团队组织。

5、决策延迟和交货时间长

当代码所有权遍及整个组织时，通常没有人是代码库中任何方面的专家。每一个变更或新功能都需要整个组织的输入。当代码和设计决策脱离团队时，它们的成本会变得很高，并且每次软件更改都会增加交付周期。

何时使用

团队中的任何人都可以在任何时间、任何地方对代码进行任何必要的更改。

如何使用

抛弃模块个人所有制的全部观念，整个代码库由整个团队共有，任何人都可以改动任何地方。辅以代码评审、CI/CD等实践疗效更好。

参考资料

- 1、<https://www.martinfowler.com/bliki/CodeOwnership.html>
- 2、<https://dzone.com/articles/collective-code-ownership-in-agile-teams>

重构

贡献人：荆修梅 徐东伟 李虎 郑乔尹



扫码查看详情

定义

重构是一种规范的技术，用于重构现有代码体，在不改变其外部行为的情况下改变其内部结构。

它的核心是一系列小的行为保持转换。每个转换（称为“重构”）作用不大，但这些转换的序列可以产生重大的重构。由于每次重构都很小，因此出错的可能性较小。每次重构后系统都保持完全正常工作，减少了系统在重构过程中被严重破坏的机会。

实践出处

在已发表的文献中首次使用“重构”一词是在 1990 年 9 月William Opdyke和Ralph Johnson的一篇文章中。格里斯沃尔德博士论文，Opdyke 博士1992年发表的论文也使用了这个术语。

为什么

重构活动有两大类好处：

可维护性。更容易修复错误，因为源代码易于阅读并且其作者的意图易于掌握。这可以通过将大型单一例程减少为一组单独的简洁、命名良好、单一用途的方法来实现。可以通过将方法移至更合适的类或删除误导性注释来实现。

可扩展性。如果应用程序使用可识别的设计模式，则更容易扩展应用程序的功能，并且它提供了一些以前可能不存在的灵活性。

何时使用

马丁·福勒在《重构》¹一书的“2.3 何时重构”小节特地提到了重构的时机：

当我谈论重构，常常有人问我应该怎样安排重构时间表。我们是不是应该每两个月就专门安排两个星期来进行重构呢？

几乎任何情况下我都反对专门拨出时间进行重构。在我看来，重构本来就不是一件应该特别拨出时间做的事情，重构应该随时随地进行。你不应该为重构而重构，你之所以重构，是因为你想做别的什么事，而重构可以帮助你那些事做好。

所以，使用重构的时机有：

- 1、重构作为日常编程的一部分；
- 2、在添加新功能时进行重构；
- 3、在修改Bug时进行重构；
- 4、在代码复审时进行重构。

到了最后的交付期限，不进行重构。

如何使用

- 封装成员变量（Encapsulate Field）：将仅限于本类使用的变量重写成私有（private）成员变量，并提供访问方法（accessor method）。这种重构方式可以将与外部调用者无关的变量隐藏起来，减少代码的耦合性，并减少意外出错的概率。
- 提取方法（Extract Method）：意思是将大段代码中的一部分提取后，构成一个新方法。这种重构可以使整段程序的结构变得更清晰，从而增加可读性。这也对函数（Function）通用。
- 类一般化（Generalize Type）：将多个类/函数共用的类型抽象出可以公用的基类（base class），然后利用多态性追加每个类/函数需要的特殊函数。这种重构可以让结构更加清晰，同时可以增加代码的可维护性。
- 函数归父（Pull Up）或函数上移：指的是方法从子类移动到父类。
- 函数归子（Push Down）或译函数下移：指的是方法从父类移动到子类。
- 方法更名（Rename Method）：方法的命名应准确地表达其用途。

输出物

重构之后的代码

常见问题解答

何时不该重构？

有时候你根本不应该重构，例如当你应该重新编写所有代码的时候。有时候既有代码实在太混乱，重构它还不如重新写一个来得简单。

重写（而非重构）的一个清楚讯号就是：现有代码根本不能正常运作。你可能只是试着做点测试，然后就发现代码中满是错误，根本无法稳定运作。记住，重构之前，代码必须起码能够在大部分情况下正常运作。

一个折中办法就是：将“大块头软件”重构为封装良好的小型组件。然后你就可以逐一对组件做出“重构或重写”的决定。对于一个重要的遗留系统，这肯定会是一个很好的方向。

另外，如果项目已近最后期限，你也应该避免重构。我们可以把未完成的重构工作形容为“债务”。很多公司都需要借债来使自己更有效地运转。但是借债就得付利息，过于复杂的代码所造成的维护和扩展的额外成本就是利息。你可以承受一定程度的利息，但如果利息太高你就会被压垮。把债务管理好是很重要的，你应该随时通过重构来偿还一部分债务。

如果项目已经非常接近最后期限，你不应该再分心于重构，因为已经没有时间了。不过多个项目经验显示：重构的确能够提高生产力。如果最后你没有足够时间重构了，这通常就表示你其实早就该进行重构了。

探针

贡献人：王用涛 晏瑞宇 徐东伟



扫码查看详情

定义

探针（Spike）是一个以回答问题或收集信息为目的的任务，而不是一个产品的交付任务。

有时在编写用户故事时，因为一些技术问题或者设计问题，团队无法很好地对用户做出估算，这时可以创建一个探针（Spike），开发团队可以做一些实际工作来探索这些技术问题或设计问题。探针的目的是找到解决问题的方法或者寻找解决问题的答案。

实践出处

探针(Spike)这个术语来自极限编程(XP)。一个Spike指的是一个用来探索/寻找潜在的解决问题的方法。

“我对2001/2002年早期的XP Universe会议的回忆是，探针一词来源于攀岩的一个类比。攀登时，我们可能会停下来将钉子钉入岩壁，来做试探，看这条路线适不适合。这种试探并不会使得我们更接近顶部，但是可以帮助我们为未来的攀登做好准备。”——Jigish Chawda

为什么

探针可以帮助我们打破不确定性。

如果有些东西没有定义好，会在实施的时候压力倍增。

探针可以帮助我们时刻明确前进的方向。

探针可以避免因为不确定性而过高的估算用户故事。

何时使用

当开发团队有很多种方法可供选择时，他们需要做一些尝试看来决定哪一种方法才是最合适的。

当开发团队不确定他们现在想到的解决方案是不是能达到预期的结果时。

当开发团队不知道怎么来解决问题时。

当开发团队要开始做一点实际的工作来估算一个或者多个用户故事时。

如何使用

- 1、在你的待办事项列表中创建一个代表探针的工单。
- 2、将这个探针加到当前的冲刺中，估算完成这个探针所需要花费的工作量。
- 3、在冲刺中完成必要的探索和设计，以便于可以估算探针对应的用户故事的工作量。
- 4、关闭这个探针并在它对应的用户故事上更新新的估算。

注意事项：

- 一直调查一个问题是不对的，要给自己加时间盒。
- 做探针时要留给自己足够的时间来获得结果，不要操之过急。
- 在做探针的时候不要开始实现用户故事。
- 始终对你现在研究的探针保持专注。
- 有时可能研究的主题是全然未知的，可以先做一些细化，然后对其中的一个具体的问题进行探针。
- 做探针时，为了估算的准确，应该进行深层次的研究而不是仅仅停留在表面。
- 不要一个冲刺里面全是探针。

输出物

一个解决潜在问题的方法或者关于一个或者多个用户故事更准确的估算。

参考资料

- 1、<https://blog.csdn.net/seagal890/article/details/102102103>
- 2、How Agile Spikes help to Improve your Agile Product Delivery? --- Anca Onuta
- 3、Agile 101: What is a Spike ? Morgan Folsom

静态代码分析

贡献人：宋晓娜 晏瑞宇 徐东伟 巨新龙



扫码查看详情

定义

静态代码分析又称静态程序分析，指在不实际执行代码程序的情况下，进行代码分析的方法。与之对应的是动态代码分析，即在代码程序运行的时候所执行的代码分析方法。

在信息安全的领域中，静态程序分析会称为静态应用程序安全检测（Static Application Security Testing，简称SAST）。

在大多数情况下，静态代码分析指的是配合静态程序分析工具进行的分析，人工进行的分析一般称为程序理解或代码审查。

实践出处

在软件开发发展之初，人们并没有意识到代码审查的必要性，大约在二十世纪七十年代，人们真正意识到了代码审查工作与生产力以及产品质量具有同样的重要性，并将之应用于软件项目开发。通过这种新生的软件开发方法能够在项目开发的早期发现并消除项目的代码缺陷，从而生产出更加可靠且高效的程序。

为什么

通过代码静态分析可以在不进行程序执行的情况下彻底分析所有代码。由于这一特性，我们可以通过静态代码分析来对深层次的、无法人工顾及的代码进行检测、监管。执行静态代码分析的另一个好处是可以根据项目的特定需求对其进行修改，并且可以轻松实现跨团队协作。此外，通过静态代码分析可以在整个开发周期的早期阶段及时发现错误，从而大幅度减少由于代码修复、优化而带来的成本。

何时使用

静态代码分析可前置，寻找可集成在IDE中的静态代码分析工具插件，便于在本地代码中发现问题，提早改善。

静态代码分析可以发生在软件开发周期的早期，一般在软件测试开始之前进行；对于实施DevOps的组织，静态代码分析应该发生在“构建”阶段。

如何使用

大部分的静态代码分析的对象是针对特定版本的源代码，也有些静态代码分析的对象是目标代码。从工具使用角度来说，大致有以下步骤：

- 1、开发者完成代码编写工作并提交代码；
- 2、通过静态代码分析工具进行代码检测，静态代码分析工具会根据预定义的或者标准化的代码检测规则来判断目标代码是否符合检测标准，并给出检测结果；
- 3、开发者根据检测结果进行代码修复工作并将修复的代码进行提交。

注意：

1. 在某些情况下，静态代码检测工具可能出现误报的情况，因此，需要通过人工对检测结果进行检查以判断是否所有检测出来的问题都需要进行修复。
2. 推荐做法：结合业务特性和公司对代码的质量要求标准，持续优化代码规则库，最终形成适合公司各研发产品线的静态代码规则库。

输出物

静态代码分析报告

参考资料

- 1、<https://www.techtarget.com/whatis/definition/static-analysis-static-code-analysis>
- 2、https://en.wikipedia.org/wiki/Static_program_analysis
- 3、<https://www.appknox.com/blog/static-code-analysis>
- 4、《Secure Programming with Static Analysis》by Brian Chess / Jacob West
- 5、<https://paginas.fe.up.pt/~ei05021/TQSO%20-%20An%20overview%20on%20the%20Static%20Code%20Analysis%20approach%20in%20Software%20Development.pdf>

代码评审



扫码查看详情

贡献人：荆修梅 鲁飞 徐东伟 晏瑞宇 郑乔尹

定义

代码评审（有时称为同行评审）是一种软件质量保证活动。代码评审需要一个或多人通过查看和阅读部分源代码来检查程序，他们在实施之后或作为实施的中断进行检查，至少其中一人不得是代码的作者。除作者外，进行检查的人员被称为“审稿人”。

实践出处

历史上第一个详细研究和描述的代码评审过程被其发明者迈克尔费根称为“检查”。费根检查是在软件开发过程的各个阶段试图发现文档（例如源代码或正式规范）中的缺陷的过程。它以Michael Fagan 的名字命名，他被誉为正式软件检查的发明者。

为什么

更好的代码质量：

提高内部代码质量和可维护性（可读性、统一性、可理解性等）。

更早地发现缺陷：

提高外部方面的质量，尤其是正确性，但也发现性能问题、安全漏洞、注入的恶意软件，开发人员无需在功能发布后发现错误并争先恐后地发布补丁，而是可以在客户看到问题之前立即发现并修复问题。通过单元测试在软件开发生命周期的早期移动审查过程有助于开发人员使用新知识进行修复。当等到生命周期结束进行审查时，开发人员通常很难记住代码、解决方案和推理。静态分析是 满足业务和客户价值的一种廉价、有效的方法。

学习/知识转移：

帮助转移有关代码库、解决方案方法、质量期望等方面的知识；无论是对审稿人还是对作者，当软件开发人员在团队成员进行更改后立即审查代码时，他们可以学习新技术和解决方案。代码审查帮助初级开发人员向更高级的团队成员学习，类似于结对编程如何有效地帮助开发人员分享技能和想法。通过在整个组织中传播知识，代码审查确保没有人是单点故障。每个人都有能力审查和提供反馈。共享知识还有助于团队成员休假，因为团队中的每个人都具有某个主题的背景知识。

增加协作：

增加集体代码所有权和团结感。当团队成员一起工作以创建解决方案时，他们对自己的工作有更多的归属感和更强的归属感。作者和审稿人可以一起寻找最有效的解决方案来满足客户的需求。加强整个软件开发生命周期的协作以防止信息孤岛并保持团队之间的无缝工作流程非常重要。要成功地进行代码审查，开发人员必须建立在协作开发中具有坚实基础的代码审查心态。

寻找更好的解决方案：

为超越手头特定代码的新的更好的解决方案和想法产生想法。

遵守 QA 指南、ISO/IEC 标准：

在某些情况下，代码审查是强制性的，例如空中交通软件、安全关键软件。

何时使用

代码评审应该在自动检查（测试、样式、其他CI）成功完成之后，但在代码合并到存储库的主线分支之前进行。

如何使用

电子邮件线程

一旦给定的代码准备好进行评审，该文件就会通过电子邮件发送给适当的同事，以便他们每个人在工作流程允许时尽快进行评审。虽然这种方法肯定比更传统的技术更灵活和适应性更强，比如让五个人在一个房间里参加代码评审会议，但包含建议和不同意见的电子邮件线程往往会很快变得复杂，这是需要让最初的编码人员来进行整理。

结对编程

作为极限编程（XP）的标志之一，这种编写软件的方法让开发人员并排（至少在形象上），一起处理相同的代码，从而在他们进行时评审彼此的工作。这是高级开发人员指导初级同事的好方法，并且似乎将代码评审直接融入到编程过程中。然而，由于作者甚至合著者往往过于接近他们自己的工作，其他代码评审方法可能会提供更多的客观性。结对编程在时间和人员方面也比其他方法使用更多的资源。

即时代码评审

对于大多数开发人员来说，比XP的结对编程更舒服，即时代码评审技术是参与同行代码评审的最古老、最简单和最直观的方法之一。一旦你的代码准备就绪，只需找一位合格的同事坐在你的工作站（或去他们的工作站）并为你评审你的代码，同时向他们解释你为什么以这种方式编写代码。这种非正式的方法当然是“轻量级的”，但如果它缺乏跟踪或记录的方法，它可能有点太轻了。

工具辅助

我们把我们的最爱放在最后，因为可以说没有比通过基于软件的代码评审工具更简单、更有效的方式来评审代码了，其中一些是基于浏览器的或无缝集成到各种标准IDE和SCM开发框架中。软件工具解决了上述方法的许多限制，以清晰和连贯的顺序跟踪同事的评论和提出的缺陷解决方案（类似于跟踪MSWord中的更改），使评审能够异步和非本地发生，发出通知当新的评论出现时，原始编码员可以保持整个流程的高效运行，无需开会，也无需任何人离开办公桌进行贡献。一些工具还允许评审和修改需求文档，并且重要的是跟踪你的进度。

输出物

代码评审意见

实战案例

以下实战案例来自于麦哲思科技：

某公司是产品开发类项目，产品开发周期比较短，销量比较大，质量要求比较高。2009年11月初开始启动过程改进，导入了每日代码走查这条实践：每天下午5点到6点为固定的代码走查时间，对当天完成的代码进行走查。2人结对，一个作者，一个专家。该公司当时共有200个开发人员，70个系统测试人员。

为了确保这条措施在企业内能够推广下去，公司制定了多方位、多手段、彼此支撑的一系列措施，见下表：

类型	措施
制度	建立每日走查一小时的制度 建立每日走查的流程 定义每日走查的检查单 代码走读检查单学习考试 SQA对代码走查的质量进行抽检，识别必须复检的走读 定义代码走查发现的缺陷类型
度量	度量代码走查的效率、速率、工作量
宣传	制定代码走查的宣传标语 代码走查的度量数据定期大幅度公布，代码走查的典型人物公开宣传 根据每个人的度量数据识别出代码走查的榜样
人员培养	代码走查的典型案例采集 典型代码的走查教育 代码走查的内部培训讲师的识别
工具	分析Pclint的报警数与系统测试的Bugs、客户反馈的Bugs之间的相关性 建立Pclint的告警数在转系统测试时的阈值 函数复杂度分析工具的使用Source Monitor 代码行数的统计工具 程序流程自动分析工具的收集与研究 定义Pclint查出的哪些告警必须修改 开发了代码走读的IT系统

在推行了1个月 后，采集了如下表的度量数据。其中代码走查列是200个开发人员每天下午结对代码走查的度量数据，系统测试列是70个系统测试人员在1个月内进行系统测试的度量数据。通过2009年12月份一个月的度量数据的分析，得到这样的结论：代码走查发现缺陷的效率是系统测试的4.25倍。

	代码走查	系统测试	比值
发现的Bugs	3687	4556	0.81
发现的严重与致命Bugs	464	1511	0.31
工作量	3086.1	16062.4	0.19
检出效率(Bugs/人时)	1.19	0.28	4.25
严重与致命Bugs的检出效率(Bugs/人时)	0.15	0.09	1.67

按功能特性创建分支

贡献人：曹江辉 鲁飞



扫码查看详情

定义

针对每个用户故事或者功能（Feature/特性）创建分支，一个用户故事或者功能只有通过全部测试后，才可以合并到主干，以确保主干是一直可以发布的。

这种分支策略让开发团队更容易在功能层次上进行工作，并保证主干的可发布状态。

实践出处

《持续交付：发布可靠软件的系统方法》

为什么

按功能特性创建分支的主要原因是保证主干的可发布状态。创建分支之后，开发团队所有的功能或者特性开发都在分支上进行，不会被其他人或者其他团队打扰。这种模式对于在功能完成前，不愿意公开其代码的团队也有一定帮助。如果每个分支合并为一次提交，那么从主干的提交记录上，每一次提交都是一个完整的特性或者缺陷修复，这样的提交记录具有更完整的语义性。

何时使用

这是一种分支的策略，如果能满足相应的条件，并且真正的理解这种模式的一些约束与准则后，经过团队的充分讨论并达成一致，就可以使用这种分支模式。

应该做充分的估计，确保采用这种模式的收益远比其客观的开销更大，而且要确保在发布日期来临之际，不会引起灾难性的后果。

需要强调的一点是，按照功能特性创建分支与持续集成是相对立的，所有的原则与准则只是为了在合并的时候情况不会太糟糕。如果能从源头避免痛苦，那就会简单很多。

需要提醒的是，当采纳这种模式时，就是“在刀尖上跳舞”。

如何使用

要想这种模式有效果，需要遵从如下一些准则：

- 1、每天要把主干上的所有变更合并到每一个功能分支。
- 2、每个功能分支都应该只具有很短的生命周期，理想情况下只有几天，绝对不能超过一个迭代周期。
- 3、活跃分支的数量在任意时刻都应该少于或者等于正在开发中的用户故事的数量。在一个迭代中，除非已经把开发中的用户故事合并回主干，否则不可以创建新的分支。
- 4、在合并回主干之前，必须确保该分支已经经过了完备的测试，没有完全通过测试的用户故事不可以合并回主干。
- 5、重构必须即时合并，从而使合并冲突最小化。这个约束非常重要，也可能非常痛苦，从而也限制了此模式的使用。
- 6、技术负责人的一部分职责是保证主干的可发布性，他应该检查每一次合并，也有权拒绝可能破坏主干的合并。

在开源世界，这种模式非常有效。同时，这种模式适合那种核心团队很小且由经验丰富者组成的商业项目中。在大型的商业项目中，这种模式也可能发挥作用，但是需要注意以下几点：

- 1、代码基被合理分解为多个模块。
- 2、交付团队被分成几个小团队，每个小团队由以为经验丰富者领导。
- 3、整个团队承诺频繁地向主干提交合并，并持续集成。
- 4、交付团队不会屈从于交付压力而达成未达标准的发布决策，例如向主干合并没有经过充分测试的用户故事等。

不过采用这种模式应该谨慎，因为它与商业软件开发中的一个反模式紧密相关。这个反模式既邪恶又常见。这个反模式就是：开发人员通过分支创建特性，而这个特性分支会独立存续很长时间，同时其他开发人员也会创建其他分支。只有在接近发布时间点，所有分支才会合并回主干。

输出物

Git的代码分支

常见问题解答

1、要确保按照功能特性分支模式的有效性，最重要的事情是哪些？

首先，所有的分支应该是短生命周期的。

其次，所有分支在合并回主干之前需要经过充分的测试。

最后，分支合并需要经过review。

2、这种模式与持续集成是对立的吗？

在没有k8s等工具出现之前，持续集成的资源耗费导致了无法对每一个分支进行持续集成，所以有这种说法。如果可以对每一个分支都进行持续集成，那么这种对立也就自然消失了。

参考资料

<https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>

按发布创建分支

贡献人：曹江辉 郑乔尹



扫码查看详情

定义

基于发布的分支策略是Git分支策略的一种，也是基于主干分支开发的几乎唯一的可以接收的创建分支的情况。当创建了这个分支，也就定义了发布的版本。所有的该版本的测试和验证全部都在该分支上进行。而最新的开发工作仍然在主干上进行。

实践出处

《持续交付：发布可靠软件的系统方法》

为什么

在基于主干的分支策略中，存在一个巨大的邪恶的做法，那就是“冻结分支”。为了某一次大的发布，几天甚至几个星期内都不允许向版本库中签入代码。这违背了每日提交的最佳实践。

而通过创建“发布分支”，开发人员仍旧可以向主干分支提交代码，而在发布分支上只做严重缺陷的修复。

只有在主干上发现最严重的缺陷修复，才从主干合并到发布分支上。

何时使用

创建发布分支是有场景要求的。开发团队需要开发新的功能，而当前要发布的版本正在测试或者准备部署，同时测试团队也希望能够在当前发布中修复缺陷，但是不影响正在进行中的新功能开发。

在这样的情况下，在逻辑上将新功能的开发与分支上的缺陷修复分开是可以接受的。但要记住的是，缺陷修复必须被合并回主干。一般来说，当把缺陷修复提交到发布分支上之后，就应该立即合并回主干。

如何使用

在基于发布的分支策略中，需要遵循如下规则：

- 一直在主干上开发新功能。
- 当待发布版本的所有功能都完成了，并且希望继续开发新功能时才创建一个发布分支。
- 在分支上只允许提交那些修复严重缺陷的代码，并且这些代码要立即马上合并回主干。
- 当执行发布时，这个分支可以选择性的打一个标签（tag）。
- 不要在已有的发布分支上再创建更多的分支，所有的后续分支应该从主干创建。从已有的分支上创建分支会形成一种“梯形”结构，很难发现两个版本之间哪些代码是公共的。
- 如果发布频率达到了一周一次（或者以上）的频度，那么该策略就没有必要了。因为在这种情况下，发布一个新版本要比在已发布的分支上打补丁更容易，成本更低。

常见问题解答

1、基于发布的分支的不适用场景？

该分支策略在真正的大项目中效果并不太好，因为很难让一个大型团队或者多个团队在同一个版本上同时完成他们所有的工作。

在大项目的开发中，需要从架构上对软件进行更合理的规划，形成组件化的架构。这样每个组件都有一个发布分支，不同开发速度的团队可以按照自己的进度在创建发布分支后，在主干分支上继续开发新的功能。

2、在基于发布的分支策略中，可以进行主干合并到发布分支的操作吗？

一般来说，因为主干已经在开发新的功能了，而这些新的功能并没有开发完成，主干并不是一个严格意义上的可用分支，这时候合并到发布分支会带来不必要的问题。一个更好的方法是，将严重的缺陷在发布分支上修改，修改完成后，立刻合并回主干分支。

持续集成

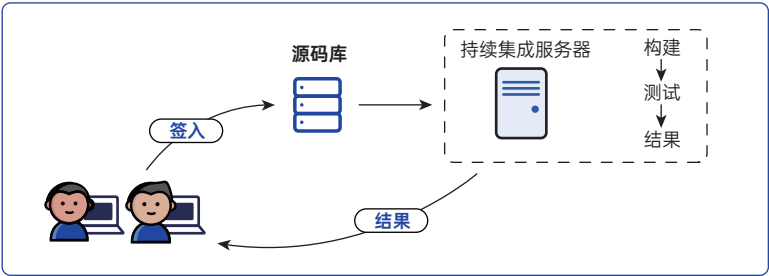
贡献人：王岩 鲁飞 晏瑞宇 徐东伟



扫码查看详情

定义

持续集成是一种软件开发实践，它促使开发团队经常将他们的更改多次集成到共享存储库。每次合并都会触发一个自动化的代码构建和测试序列。



实践出处

“持续集成”一词最早是由Grady Booch 于 1994 年创造的，出现在Booch的《面向对象的分析和设计与应用程序》一书中：“微过程的需求决定了开发团队将完成更多的内部发布，只有少数可执行版本移交给外部方。这些内部版本代表了一种系统的**持续集成**，并迫使微过程的关闭。”在1996年Kent Beck提出的**极限编程**（Extreme Programming，简称XP）中被定义，是其最初的十二种实践之一。

为什么

使用持续集成好处：

一、在处理软件部署业务方面的优势：

- 1、消除手动 FTP 部署；
- 2、预防和减少生产和测试的问题；
- 3、生成代码库健康状况的分析和报告。

二、在业务方面，持续集成的价值在于：

- 1、降低风险；
- 2、减少整个开发和部署过程的开销；
- 3、通过提供质量保证来提高公司的声誉。

何时使用

如果项目开发的规模较小，软件集成不是问题。如果项目很大，需要不断添加新功能，或不断的升级产品，则需要进行反复集成，因此必须使用持续集成来简化工作。

如何使用

一、使用CI需要遵循如下原则：

- 1、所有的开发人员需要在本地机器上做本地构建，然后再提交到版本控制库中，从而确保他们的变更不会导致持续集成失败；

- 2、开发人员每天至少向版本控制库中提交一次代码；
- 3、开发人员每天至少需要从版本控制库中更新一次代码到本地机器；
- 4、需要有专门的集成服务器来执行集成构建，每天要执行多次构建；
- 5、每次构建都要 100% 通过；
- 6、每次构建都可以生成可发布的产品；
- 7、修复失败的构建是优先级最高的事情；
- 8、测试是未来，未来是测试。

二、CI是一种理念，要落实离不开好的工具。构成有效CI的关键实践如下：

- 1、统一的代码库；
- 2、自动构建；
- 3、自动测试；
- 4、每个人每天都要向代码库主干提交代码；
- 5、每次代码递交后都会在持续集成服务器上触发一次构建；
- 6、保证快速构建；
- 7、模拟生产环境的自动测试；
- 8、每个人都可以很容易的获取最新可执行的应用程序；
- 9、每个人都清楚正在发生的状况；
- 10、自动化的部署。

三、怎么做：

- 1、开发人员将代码检出到他们的个人工作空间；
- 2、完成后，将更改提交到存储库；
- 3、CI 服务器监视存储库并在更改发生时检出更；
- 4、CI 服务器构建系统并运行单元测试和集成测试；
- 5、CI 服务器发布可部署的工件进行测试；
- 6、CI 服务器将构建标签分配给它刚刚构建的代码版本；
- 7、CI 服务器通知团队构建成功；
- 8、如果构建或测试失败，CI 服务器将向团队发出警报；
- 9、团队尽早解决问题；
- 10、继续在整个项目中不断集成和测试。

四、团队的责任：

- 1、尽可能频繁的提交代码；
- 2、不要提交不能运行的代；
- 3、不要提交未经测试的代码；
- 4、不要在构建失败时提交代码；
- 5、提交代码后不要回家，直到系统构建完成。

参考资料

- 1、Continuous Integration by Martin Fowler.<https://martinfowler.com/articles/continuousIntegration.html>
- 2、The Benefits of Continuous Integration Written by:Manuel Weiss April 11, 2013. <https://www.cloudbees.com/blog/benefits-of-continuous-integration>.
- 3、《Continuous Integration-Improving Software Quality and Reducing Risk》by Paul Duvall, Steve Matyas, and Andrew Glover 2007.

每日构建

贡献人：王婷 晏瑞宇 郑乔尹 王连杰 徐东伟



扫码查看详情

定义

每日构建（Daily build）或每夜构建（Nightly build）是每天完成最新版本程序的软件构建的做法。这样可以首先对其进行编译以确保存在所有必需的依赖项，并且可能经过测试以显示没有引入任何错误。每日构建通常也是公开可用的，允许访问最新功能以获取反馈。执行每日构建有助于确保开发人员能够合理确定地知道出现的任何新错误都是他们自己在最后一天完成的工作的结果。

实践出处

种种迹象表明，早在20世纪90年代，微软公司就使用了每日构建的做法。

下文中已经提到的“心跳”以及“脉冲”都来自1995年的微软。

1996年，Steve McConnell（代码大全作者）在《Rapid Development: Taming Wild Software Schedules》中推荐将其作为最佳实践，长期以来，它一直被认为是微软开发方法的一个功能。

2000年左右，Martin Fowler在XP中将其继续演化成持续集成。

为什么

能减少开发的混乱

对于每日构建的流程，开发人员只要负责提交代码到代码库中。不需要挨个给测试人员提供编译后的二进制文件。

能减小产品低质量的风险

在开发阶段，由于测试拿到的程序都是自动编译出来的，因此保证了所有测试人员拿到的是统一的运行程序，并且这个程序和代码库中最新的代码是相对应的，每天对集成的代码做一些冒烟测试，即可杜绝项目中那些基本的质量问题。通过这种方式，使系统达到一种周知的良好状态。

能最小化集成风险

对于每日构建来说，每天都会产生安装包（或者安装光盘的ISO镜像）。从项目开始开发的那天起，每天都在进行集成。因此，集成的问题，在一开始就会暴露出来，而不用等到项目后期。

能简单化错误诊断

当系统每天都进行build和测试时，系统任何一天发生的错误都能够变得十分精细，便于排查。比如在17日系统还运行正常，18日就出错了，那么只需要检查这两次build之间的代码变化就可以了。

能极大鼓舞项目组的士气

看到产品的不断成长，能够极大的鼓舞项目组的士气，有时甚至不管这个产品到底用来做什么。开发人员可能会为系统显示了一个矩形而感到激动。通过每日构建，产品每天进步一点点，保证项目士气的持续高涨。

何时使用

每日构建，最重要的就是“每日”。如Jim McCarthy所说，把每日构建看作是项目的“心跳”，没有“心跳”的话，项目也就死了。Michael Cusumano和Richard W. Selby描述了另外一种隐含的比喻，把每日构建比作项目的“同步脉冲”。不同开发人员写的代码在他们的“脉冲”之间肯定都会存在“同步”的差异，但是必须有这样一个“同步脉冲”，使得这些代码能够组合为一个整体。当项目组坚持每天把这些不同的“脉冲”组合到一起的时候，开发人员脱离整体的情况就会得到极大程度的杜绝。

如何使用

提交代码到源代码管理服务器

编程人员实现了某个功能点之后，把代码统一提交到代码控制中心。

从源代码服务器取出代码到编译服务器

可通过编写对应的脚本文件，可通过系统的计划任务调用脚本，让脚本自行执行，不需要人为干预。

在编译服务器生成最终安装包

可通过脚本把所有需要编译的模块都编译过，然后再组装成安装包。

提交安装包到发布服务器

把编译好的安装包自动传输到发布服务器上。

从发布服务器获取安装包进行冒烟测试

编辑和构建完成后，必须对系统做基本的冒烟测试，这种测试强调的功能覆盖率，而非对功能正确性做验证。

测试人员从发布服务器上获取安装包并测试

通过基本冒烟测试后测试人员从发布服务器获取到安装包后，可对系统做更为详细的功能验证测试。

输出物

最终生成的产出物必须和交付到用户手中的一样（比如你最终提交给用户的是一个安装程序，那就必须在开发过程中每天编译出一个安装包）

参考资料

- 1、https://en.wikipedia.org/wiki/Daily_build
- 2、<https://stevemcconnell.com/articles/daily-build-and-smoke-test/>
- 3、<https://martinfowler.com/articles/originalContinuousIntegration.html>

测试

测试过程

测试分类

缺陷报告

贡献人：朱美玲 王磊 徐东伟



扫码查看详情

定义

缺陷：总的来说就是软件系统不满足用户需求，功能实现错误，功能实现遗漏，功能实现多余，测试人员认为难以理解，影响用户体验等问题。

缺陷报告：测试缺陷报告也叫Bug单，在执行测试用例时，发现软件缺陷之后就需要输出此文档，属于测试工作的重要内容，一般会用缺陷管理工具进行Bug的追踪和管理。

为什么

报告软件的缺陷非常重要。这是因为要启动缺陷修复流程，开发人员就需要了解出现了问题，并且还要了解问题是什么。如果开发团队不了解这些，那么问题存在的时间会更长。

报告缺陷还可以防止将来出现更严重的问题。通过让开发团队了解一个小错误，就可以帮助防止出现其他相关问题。

何时使用

发现缺陷后一般要先分析缺陷，排除干扰因素，确认复现缺陷的准确步骤，记录缺陷产生的环境和条件等，最后输出缺陷报告。

如何使用

缺陷报告主要由以下部分组成：

- 缺陷ID：唯一标识缺陷的字段，一般缺陷管理工具自动分配；
- 标题：简短、准确，提供缺陷的本质信息，尽量使用关键词，便于搜索；
- 基本描述：

◦ 测试环境	◦ 测试数据	◦ 操作步骤	◦ 期望结果	◦ 实际结果
--------	--------	--------	--------	--------
- 严重性：致命、严重、一般、微小；
- 缺陷类型：如功能、UI、性能、文档；
- 缺陷产生可能性：可再现的概率；
- 缺陷来源：需求、设计、编码；
- 产品模块：缺陷出现在产品的哪个模块中；
- 版本：缺陷出现在产品的哪个版本中；
- 缺陷原因：数据格式、计算错误、接口参数、变量定义与引用等；
- 附加信息：Trace log、图片、操作过程的视频等；
- 缺陷状态：缺陷的活动状态。

- 新提交的（New）
- 已分配的（Assigned）
- 待解决的（Open）
- 已修改的（Fixed）
- 问题未解决的（Reopened）
- 已归档的（Closed）

缺陷的严重性：衡量缺陷对客户满意度的影响程度

- 致命的/fatal：致命错误，死机、丢失数据，数据溢出错误；
- 严重的/critical：严重错误，较大的功能问题；
- 一般的/major：一般的错误，不影响基本功能的使用；
- 微小的/minor：界面美观等等比较小的问题。

缺陷修复优先级：缺陷被修复的紧急程度

- 立即解决（P1）：缺陷导致系统几乎不能使用或测试不能继续，需立即修复；
- 高优先级（P2）：缺陷严重，影响测试，需要优先考虑；
- 正常排队（P3）：缺陷需要正常排队等待修复；
- 低优先级（P4）：缺陷可以在开发人员有时间的时候被修正。

这些是Bug单比较重要的部分，当然还有其他部分，如发现人、发现时间、指派给等，一般公司都会有具体的模板和要求，不同的缺陷管理工具对于缺陷的严重程度和优先级定义略有不同。

输出物

缺陷报告单

参考资料

- 1、<https://cloud.tencent.com/developer/article/1728039>
- 2、朱少民《软件测试方法和技术》第3版

测试计划

贡献人：徐东伟



扫码查看详情

定义

可以把测试计划视为对测试过程所做的“项目计划”。

测试计划可以看作是测试工作的说明手册或指南。它描述了：

- 测试的目标（计划验证和/或确认什么）；
- 测试范围（测试和不测试什么）；
- 想要执行的活动的的大体的（有时是详细的）时间表（想要什么时候以及如何测试）。

测试计划应列出项目中预见的风险及其各自的级别，以便根据风险对测试进行优先级排序。

测试计划中最重要的部分可能是所需资源的定义。资源可以被看作是由人（例如参与测试的人）和技术（例如测试环境、测试工具和测试数据）组成。

为什么

无论项目生命周期方法如何，测试计划都是测试的基本活动。测试计划就像测试的项目计划。

在测试的许多方面，需要一定程度的计划和准备，以便在需要时将所需资源到位。某些资源，例如人员和环境，可能需要大量准备。测试计划是定义这些资源和表达测试需求的地方。

测试计划的一个主要目标是与组织的其他部门（可能还有其他组织）就如何计划进行测试进行沟通。如果没有测试计划，关于测试的交流会变得非常活跃，人们可能在任何给定时间都不知道测试的目标和期望。

何时使用

无论采用哪种生命周期方法，测试计划都是一种有价值的工具，可确保提供适当的资源来满足测试目标。

- 创建测试计划的一个很好的起点是定义测试策略。软件测试策略有助于理解测试的广泛目标以及特定项目或版本的独特之处。在制定完测试策略后，就可以开始创建测试计划了。

如何使用

编写测试计划通常是测试经理或领导的职责。测试团队和组织中的其他人（如用户和开发人员）可以为测试计划提供输入以及评审测试计划，但通常由测试经理来实际编写测试计划。

在测试计划的初稿中通常存在差距和模糊性。很多时候，测试计划中所需的信息会随着时间的推移而出现。事实上，可能有一些测试细节直到测试前不久才变得清晰。

编写测试计划时，你会发现，当你试图了解计划中所需的细节时，写测试计划的工作就变成了一项调查工作。一个好的做法是，将测试计划的某些部分分配给测试团队的成员去调查并记录下来。测

试经理作为测试计划的作者，可以对收集过来的信息进行编辑，从而形成最终的测试计划。

在创建测试计划时，最重要的任务之一可能就是对其进行评审。第一次评审应该是团队评审，邀请了解待测内容的测试团队成员参与。

在进行任何必要的更改之后，下一次评审应该邀请具有丰富知识的利益相关者，例如项目负责人、测试团队负责人、技术测试分析师、业务分析师、主题专家和任何其他可以在评审中提供有用观点的人。

在编写测试计划时，请注意，只有部分测试计划细节涉及大量基于技术细节的信息。测试计划的其余部分要包含所有利益相关者（无论是什么角色）都应该易于阅读的信息。

测试计划应满足如下特点：

- 简洁：语句简明扼要，最好是用列表的形式。
- 结构良好：开头做总体介绍，然后在计划的正文中提供更详细的信息。使用测试计划的标准模板，有助于对测试计划的内容进行组织。对章节和主题进行编号，有助于未来进行引用。
- 易读性好：使用大多数读者都能理解的简单语言。
- 适应变化：测试计划要随着实际情况的变化而即时进行更新和修改。
- 准确：人们应该能够相信测试计划中包含的信息是准确的。如果发现错误，应尽快报告并更正。

输出物

测试计划

参考资料

- 1、<https://www.practitest.com/qa-learningcenter/best-practices/write-a-test-plan/>
- 2、<https://www.practitest.com/assets/pdf/system-integration-testing-template.pdf>

测试策略

贡献人：徐东伟



扫码查看详情

定义

测试策略是描述软件产品测试方法的文档。创建测试策略是为了告知项目经理、开发人员和测试人员测试过程中的关键问题。

测试策略描述了如何在测试层级缓解风险、需要应用什么标准，以及需要执行的测试类型。

编写测试策略是一项随着经验而发展的技能。

测试策略需要在整个项目团队中传播，以便团队成员在测试方法和职责上保持一致。

为什么

在软件行业，一个产品在发布之前都需要经过测试。在测试产品之前，需要一些计划和方法。一个质量工程师团队坐在一起，决定需要对产品进行哪些类型的测试，才能够证明要发布的产品是高质量产品。

为此，他们决定某种类型的策略，描述测试什么、如何测试以及何时测试。所有这些都需要记录在某处。这就是测试策略文档的由来。这是测试过程的第一份重要文件。编写一个好的测试策略文档可以改进整个测试过程。测试策略文档能够为我们后续的测试计划的制定的书写提供指导。

何时使用

对于开发设计的每个阶段，应创建相应的测试策略来测试新的功能集。

如何使用

在编写测试策略时，在决定包含哪些内容时，应考虑组织和项目的不同情况。但是，测试策略文档中通常包含以下内容：

- 测试层级：解释该文档是为哪个测试层级而写的，可以是单元测试、集成测试、系统测试或验收测试。
- 范围：列出要测试哪些部分。并且还要明确提及哪些部分不在测试范围内。
- 进入和退出条件：应该定义测试开始和结束分别要满足什么样的特定条件。此信息可能来自产品经理或其他利益相关者。
- 环境：描述测试发生的地方。它是在开发、测试、准生产还是生产环境中？如果涉及数据库，它指向哪个环境？确保列出所有 URL。
- 团队责任：描述谁负责执行每项任务，这个描述可能包括来自测试团队之外的人。例如，开发人员可能负责单元测试，或者利益相关者可能负责验收测试。测试人员可以执行脚本，测试经理可以负责报告。
- 测试工具：这部分应该定义测试和测试用例管理所需的任何工具。
- 技术：解释如何定义条件和测试用例。是否需要创建测试用例数据？如果需要，请说明需要什么样的数据以及什么时候需要。
- 缺陷管理：解释如何报告和跟踪Bug。你应该概述如何定义Bug的严重程度，并描述解决Bug的流程。

- 发布控制：定义谁将负责发布。发布会发生在迭代中，还是持续发布（CI/CD）？发布前需要哪些审批？如果有评审委员会流程，请务必包含该信息，因为它会影响发布时间。
- 度量指标和报告：描述记录度量指标的方法以及谁负责报告。你可能需要产品经理就每个指标的可接受值提供意见输入。
- 风险和缓解：你可能在测试之前就已经意识到威胁。如果可能，请描述降低风险的方法。
- 概述：包括对测试及其产出物的基本概述。

收集完所有这些信息后编辑成文档。网上有很多模板，或者作者可以自己创建。如果你的组织还不能熟练书写测试策略文档，我这里要告诉你的是，在你设计出适合你的组织或项目的模板之前，需要对模板调整好多次。

你有很大的自由来编写测试策略。尽管确实存在测试策略标准 (ISO/IEC/IEEE 29119-3)，但您仍然可以自己制定。你的测试策略可以只有一页，并且创建时间不到一个小时。

输出物

测试策略文档

参考资料

- 1、<https://blog.testlodge.com/what-is-test-strategy/>
- 2、<https://blog.testlodge.com/wp-content/uploads/2022/06/test-strategy-example-pdf.pdf>

回归测试

贡献人：Becca



扫码查看详情

定义

回归测试是一种测试类型，是当产品代码发生变化后，重新对产品进行测试，以确认代码变化是否对现有功能造成损害。代码变化包括：新功能、缺陷修复、配置改变、代码重构等等。回归测试使用部分或全部已执行过的测试用例，重新进行测试，以保证产品仍能正常运行。

由于任何代码修改都可能引起问题，所以在软件开发的各个阶段都需要进行多次回归测试，并且回归测试用例集随系统功能增加而增加，这些因素让回归测试变得成本巨大，测试时间也越来越长。因此，回归测试最好是自动化测试，并且需要选择合适的回归测试策略来改进回归测试的有效性和效率。

在敏捷开发中，短周期频繁发布新版本需要频繁的完成回归测试。在极限编程方法中，甚至要求每天都进行若干次回归测试。

为什么

只要代码发生了改变，就可能给带来问题，并且可能会对关联的很多模块甚至整个系统造成影响。所以需要回归测试来重新验证已有的功能，以确认代码的变化是否损害原有的正常功能。

何时使用

在软件生命周期中的任何一个阶段，只要发生了代码改变，无论是新功能、缺陷修复、配置改变、代码重构等，都可能带来问题，都需要回归测试。

但考虑成本和效率，不一定每次都需要/来得及做充分的回归测试。

通常在产品迭代发布之前需要全面的回归测试以保证上线质量。

在瀑布开发中，一般是周期性进行回归测试。一般到了项目快要交付的时候，对代码改动的控制极其严格，任何代码改动都要做回归测试。

在敏捷开发中，每个迭代产生的增量都是潜在可交付的，也就都需要通过回归测试。使用持续集成工具和自动化测试可以更好的支持短周期迭代的频繁的回归测试。

如何使用

大致步骤

- (1) 制定回归测试策略（范围、频率、触发条件等）；
- (2) 生成测试用例库；
- (3) 识别出软件中有修改的部分；
- (4) 基于回归测试策略，从基线测试用例库中，选择适用的测试用例集；
- (5) 执行回归测试，生成测试报告；
- (6) 别忘了更新基线用例库，添加必要的新用例，并去除或优化旧用例。

回归测试策略

回归测试策略需要兼顾有效性和效率。常用策略包括：

(1) 全量回归测试策略

全量回归是选择基线测试用例库中的全部测试用例，这是一种安全系数最高、风险等级最低的策略，但测试成本最高。全量回归测试虽然有效，但不高效。

(2) 非全量回归测试策略

随着功能增加，测试用例会越来越多，回归测试的工作量越来越大。在时间和质量等各种条件下，选择非全量回归测试策略，使用部分测试用例进行回归测试。

这种策略的核心思想就是测试用例集的选取和优化，提高测试效率，减少资源投入。

但是这种策略有漏测风险，应该结合基于业务风险驱动，从业务关键度与故障可能性这两个维度的风险分析结果采用相应的测试用例集进行回归测试。

- 基于风险：从业务风险和技术风险两方面考虑，基于一定的风险标准来从基线测试用例库中选择回归测试集。首先运行最重要的、关键的和可疑的测试，而跳过一些非关键的、优先级别低的或者高稳定的测试用例。
- 基于操作剖面（操作主流程）：如果基线测试用例库的测试用例是基于软件操作剖面开发的，测试用例的分布情况反映了系统的实际使用情况。回归测试可以优先选择那些针对最重要或最频繁使用功能的测试用例，释放和缓解最高级别的风险，有助于尽早发现那些对可靠性有最大影响的故障。
- 基于代码修改的关联关系：在进行局部修改时，如果对局部化有足够的信心，可以通过相依性分析修改的影响范围，将回归测试局限于被改变的模块和它的接口、以及关联功能等。回归测试尽可能覆盖受到影响的部分。
- 基于指标：先确定一个要达成的指标，如系统的性能指标、所修改代码的覆盖率，与修改有关的接口的覆盖率等，基于这些指标选择最小的回归测试用例集。

回归测试用例集

当得到一个软件的基线版本时，用于基线版本测试的所有测试用例就形成了基线测试用例库。回归测试的用例可以根据回归测试策略从基线测试用例库中选取合适的测试用例组成回归测试集。

为了保证测试用例库中测试用例的有效性，必须对测试用例库进行维护，删除或优化旧用例，追加新功能的新用例。测试用例的维护是一个不间断的过程，需要持续进行。

回归测试集的选择

在软件生命周期中，即使一个得到良好维护的测试用例库也可能变得相当大，这使每次回归测试都重新运行完整的测试集变得不切实际，所以需要基于回归测试的策略，选择一个缩减的回归测试集来完成回归测试。回归测试策略考虑业务和技术角度的风险和优先级，使用代码相依性分析等安全的缩减技术，尽量平衡回归测试的有效性和效率两个方面。

可以参考以下方面选择回归测试用例：

- 经常出现缺陷的测试用例；
- 用户更容易看到/使用到的功能；
- 验证产品核心特性的测试用例；
- 经历了更多和最近的变化的一功能的测试用例；
- 所有集成测试用例；
- 所有复杂测试用例；
- 边值测试用例；
- 成功测试用例；
- 失败测试用例。

自动化回归测试

由于回归测试是一个反复的过程，并且大量重复相同测试用例，使用自动化的方式是最高效的。

需要从业务流驱动、数据与用例分离、组件技术、统一的平台等多维度出发，进行长期建设。

回归测试常用工具：

Selenium：是用于自动化Web应用程序的，可用于基于浏览器的回归测试。

QTP（Quick Test Professional）：是一个自动化功能和回归测试用例的自动化软件。

Robot Framework等等。

输出物

回归测试策略

回归测试用例集

回归测试结果

常见问题解答

1、每次代码提交都要进行回归测试吗？

不一定，回归测试的触发条件和测试范围需要根据测试策略来设定。

使用敏捷开发的持续集成和每日构建实践，可以支持更频繁的自动化回归测试，又不会增加太多成本。

2、缺陷修复后的重新测试(Bug fix验证) 和 回归测试的区别？

重新测试是针对缺陷修复后的，即当开发修复了缺陷后，需要再次测试此处功能或错误以确保代码确实已修复。

重新测试的范围是在缺陷本身。

而回归测试并不用于验证缺陷修复。回归测试虽然也会被缺陷修复所触发，但范围是更大更全面的，以确保修所改变的代码不会影响软件的其他部分。

参考资料

1、百度百科 - 回归测试

2、Regression-Testing

3、知乎 - 什么是回归测试

4、知乎 - 大佬眼中的回归测试

系统集成测试



扫码查看详情

贡献人：Becca

定义

系统集成测试，简称集成测试，是将已通过单元测试的组件作为输入，将它们采用一次性或增量方式集成为更大的组件或整个系统，验证系统的接口、数据、功能等各方面是否符合设计的要求，验证组件之间的连通性、可用性、联动性，逻辑性等，发现集成系统的问题并返回给开发团队进行修改和完善。

集成测试一般有模块级、子系统级、系统级。

当系统的所有组件都集成在一起并通过系统集成测试后，这个集成系统就作为输出进行下一阶段的系统测试。

实践出处

集成测试是软件开发V模型中的一个测试阶段。V模型把测试分为：单元测试、集成测试、系统测试、验收测试。

为什么

只要代码发生了改变，就可能给带来问题，并且可能会对关联的很多模块甚至整个系统造成影响。所以需要回归测试来重新验证已有的功能，以确认代码的变化是否损害原有的正常功能。

何时使用

大量实践表明，有时单元组件虽然可以单独工作，但是并不能保证组装起来也可以同时工作。例如接口不统一无法集成；数据可能在通过接口的时候丢失；功能逻辑和流程可能不通顺或不合理；模块之间的互相作用产生了没有预期到的问题等等。所以集成测试是必不可少的，系统集成测试也是后续进行系统测试和验收测试的必要的前提条件。

当系统非常复杂时，比如需要由多个模块或子系统互相联动才能完成核心功能；跨学科、跨领域的技术；多重异构、分布式、软硬件结合；由不同团队协同开发等等，更加需要集成测试来尽早发现集成系统的问题。

- 更早进行测试，更早获得反馈
 - 完成单元测试的组件就可以进行集成测试。
 - 在集成测试发现了缺陷马上送回开发团队，查错和修复的成本越低。
 - 可以更早的验证集成的模块是否按预期工作。
 - 集成的模块或系统更方便用户进行早期反馈。
- 扩大验证范围，提高验证效率
 - 集成测试覆盖多个模块或子系统，测试效率更高。
 - 重点关注组件的集成交互，接口测试易于自动化，性价比高。
 - 持续集成让集成的风险更小，问题排查更清晰，质量随时受控并处于可接受状态。

当软件的单元组件完成了单元测试之后，就需要集成到更大的组件中进行集成测试。直至整个系统的组件都集成在一起。

在敏捷开发中，推荐持续集成的实践，每当有新开发的功能模块代码合并到主分支代码时，就需要进行集成测试，并且通常是自动化测试。

如何使用

1、进行被测系统的结构、模块、接口等分析。

2、确定集成测试的策略。例如：

- 非增量集成测试策略：大爆炸式集成。
- 增量的集成测试策略：自顶向下、自底向上，混合式(三明治式)、基干式（从核心功能到外围）、基于风险，基于流程等等。

3、确定集成测试的范围和关注重点，例如：

- 接口是否一致（各种数据模型，输入输出方式，微服务接入方式等）；
- 数据流和控制流是否按照设计实现其功能；
- 集成后的功能可用性、正确性；
- 集成后的性能、稳定性、兼容性等；
- 集成后的联动逻辑性、合理性、可操作性等；
- 如果是微服务架构的系统，集成测试方法主要是接口API测试。

4、生成测试计划、编写测试用例、搭建测试环境等（敏捷开发中需要使用持续集成工具平台）。

5、执行集成测试。

6、生成集成测试报告并尽快反馈给开发团队。

输出物

集成测试策略、计划、测试用例集、测试环境、集成测试结果报告等。

敏捷开发中，还要搭建能进行持续集成测试的工具或平台。

常见问题解答

系统集成测试与系统测试的区别

1、测试的对象不同：

集成测试的对象是模块或子系统的集成和调用关系，主要是针对程序内部结构进行测试，包含接口、数据、功能连贯性等。

系统测试对象是整个系统，将系统中的硬件、软件、操作人员看作一个整体，来验证系统功能是否正确实现、是否遗漏，性能、安全等是否符合要求。

2、测试的角度不同：

集成测试更多从系统的结构和逻辑进行测试，验证系统可以按设计运行。

系统测试除了考虑符合设计，还要从用户使用的角度进行测试，验证系统可以按照用户的期望执行（功能、性能、操作习惯等）。

3、测试的依据不同：

系统集成测试是验证集成系统符合《设计说明书》。

系统测试是验证系统满足《需求规格说明书》《产品使用说明书》等。

4、测试的方法不同：

集成测试一般采用灰盒（白盒加黑盒）的方式来测试。

系统测试一般采用黑盒方式来测试。

参考资料

1、《敏捷测试》

2、《Continuous Delivery 持续交付》

3、wikipedia - system integration testing

4、system-integration-testing

5、百度百科 - 集成测试

6、百度百科 - 系统测试

7、CSDN - 集成测试

实战案例

采用微服务架构，增量式的集成测试策略，使用敏捷开发和DevOps工具，在持续集成平台上进行频繁和持续的自动化集成测试。

欢迎补充更详细的实战案例。

用户验收测试

贡献人：Becca



扫码查看详情

定义

在软件开发的V模型中，用户验收测试是在开发完成后、交付给客户投入实际应用之前的最后一个测试阶段，是由最熟悉业务的用户在预生产或生产环境中进行验证，确定产品是否能够满足需求规格说明书、能否满足合同中规定的各项要求，用户能否接受。（正式验收之前，通常会先进行Alpha测试、Beta测试）

验收测试分为功能测试和非功能测试。非功能测试一般包括安全性、易用性、可用性、可扩展性、兼容性、效率、资源占用率、用户文档等方面。

验收测试是面向业务需求的。用户验收测试由用户执行，以用户视角使用产品，而不是以开发人员所认为的正确方式来运行程序。用户验收测试验证产品确实实现了客户的真正需求。Alpha测试、Beta测试会邀请专门的客户进行测试。灰度发布是通过大量真实用户进行验收测试。

在敏捷开发中的验收测试是基于用户故事的。

用户故事是需求承载的最小单元。在迭代中开发的用户故事都需要有验收标准AC（Acceptance Criteria），指定验收场景和标准。根据用户故事INVEST原则可以知道，用户故事是独立的、可验证的，也就是说，是可以被独立验收的。所以不需要等到全部代码开发完成才能做验收，而是基于每个用户故事的AC来验收。

用户故事只有通过了所有的验收测试，才能被视为完成。也就是说，每次迭代都要创建验收测试，否则开发团队就不会有任何进展。

验收测试最好是自动化的，并收入其他测试的测试用例库，例如回归测试。

实践出处

验收测试是软件开发的V模型中，软件开发和测试过程的最后一步。

验收测试通过，产品就可以交付给客户了。

为什么

在执行了单元、集成、系统测试之后，系统已经能正常运行了，为何还需要用户验收测试？

在用户验收测试之前，所有测试都是由开发人员或测试人员根据需求设计，用自己的知识解释需求并进行测试。

有可能会出现这样的情况，虽然开发的功能是符合需求文档的，但有一些只有最终用户了解的业务需求细节和流程可能没有被覆盖到，或者被误解了。

另外，之前的测试都是使用开发环境中的数据。而验收测试可以使用最接近真实的用户数据和真实用例。也很容易出现之前没有遇到的问题。

所以用户验收测试是非常必要而且关键的。

在敏捷测试中，更是重视用户验收，对每一次短周期迭代都进行验收测试。缩短反馈环，减少缺陷修复的成本。

何时使用

验收测试是软件开发的V模型中软件开发和测试过程的最后一步。

经过了单元测试、集成测试、系统测试的软件，需要进行用户验收测试。用户验收测试通过，产品就可以交付给客户使用了。这是项目发布周期非常关键的环节。

在瀑布开发中常见的验收测试环节：

- Alpha测试：Alpha可以认为是内测版，一般在项目中后期。
- Beta测试：Beta可以认为是公测版，一般在项目中后期。
- 最终的用户验收测试：在项目交付之前进行。

在敏捷开发中：

- 在每个短周期迭代结束时，对每个完成的增量（用户故事）都会进行实时的验收测试。

如何使用

瀑布模型中的验收测试，以Beta测试为例。常见流程：

- 1、确定参加beta测试的用户。一般是邀请重要客户或合作伙伴参加。
- 2、用户提供业务场景和测试数据，贴近用户实际应用环境和使用方式。
- 3、与用户确定测试策略和测试计划。
- 4、准备用户培训手册，例如产品的安装文档，使用说明，功能手册等。
- 5、准备测试脚本：测试步骤详细描述文档，自动化测试脚本等。
- 6、组织用户进行测试。
- 7、生成测试结果报告，评审分析结果，反馈发现的问题等。

在敏捷开发中的验收测试：

当每个短周期迭代完成，一般在评审会时，会对每个完成的用户故事都会经过内部验收甚至客户验收。依据是用户故事的验收标准AC。

可以使用ATDD/BDD，将验收条件变成可执行的测试用例，进行自动化验收测试。

验收测试用例可以作为后续的回归测试用例，以保证验收过的功能一直符合要求。

参考资料

- 1、《敏捷测试 - 以持续测试促进持续交付》
- 2、wikipedia - acceptance_testing
- 3、百度百科 - 验收测试

单元测试

贡献人：Becca



扫码查看详情

定义

单元测试是指对软件中的最小可测试单元进行检查和验证，用于测试一个很小很明确的功能是否正确，可以第一时间发现模块内部存在的各种错误。对于单元的含义，是根据实际情况人为规定的最小的被测代码段，例如一个函数、一个类、一个算法模块、一个图形等。

单元测试关注代码的内部逻辑，进行功能项、数据项的正确性、完善性测试，并考虑多种分支条件、正常和异常处理等多个角度进行验证，并关注单元接口的正确性。

单元测试的创建简单、易于维护、易于自动化、方便执行、具有回归性，还可以加入到后续其他测试的用例集中使用。

单元测试是白盒测试，由代码开发者编写和执行。如果发现问题就可以立刻修改。独立的单元可以在与程序的其他部分相隔离的情况下进行测试。

在敏捷开发中，团队的“完成的定义DoD”中通常会包含单元测试：“必须完成单元测试，并且覆盖率达到某合理范围，才能算开发完成”。

经常与单元测试联系起来的另外一些开发活动包括代码评审（Code Review），静态分析（Static Analysis）和动态分析（Dynamic Analysis）。静态分析就是对软件的源代码进行研读，查找错误或收集一些度量数据，并不需要对代码进行编译和执行。动态分析就是通过观察软件运行时的动作，来提供执行跟踪，时间分析，以及测试覆盖度方面的信息。

实践出处

单元测试是软件开发V模型中测试活动的第一个环节。

为什么

1、单元测试可看作是编码工作的一部分。执行单元测试，就是为了证明这段代码的行为和期望一致。只有完成单元测试的代码才能算作已经完成。

2、实践表明充分的单元测试可以在软件开发的早期发现很多的Bug，并且修改它们的成本也很低。在软件开发的后期阶段，Bug的定位和修改将会变得更加困难，并要消耗大量的时间和成本。无论什么时候作出修改都要进行回归测试，在生命周期中尽早地对软件产品进行测试将使效率和质量得到最好的保证。经过测试的单元组件会让系统集成过程更加顺利和简化，开发人员可以将精力集中在集成的交互作用和全局的功能实现上，而不是陷入充满很多Bug的单元之中。

3、《实用软件度量》(Capers Jones, McGraw-Hill 1991)列出了准备测试，执行测试，和修改缺陷所花费的时间（以一个功能点为基准），这些数据显示单元测试的成本效率大约是集成测试的两倍、系统测试的三倍。所以单元测试是性价非常高的测试活动。

4、有充分的单元测试的保护，才可以更安全 and 高效的重构代码。

何时使用

单元测试是软件开发V模型的测试阶段第一步。开发代码的同时就编写单元测试用例，并进行单元测试。

单元测试越早越好，早到什么程度？极限编程（Extreme Programming，或简称XP）讲究TDD，即测试驱动开发，要求先编写测试代码，再进行开发。详情请见TDD相关实践。

如何使用

1、常见的单元测试方法

准备，执行，断言（Arrange, Act, Assert）

- 准备测试环境,测试数据等；
- 执行被测试方法；
- 用断言来验证执行结果。

2、单元测试用例设计

单元测试一般是白盒测试，会针对程序逻辑结构设计测试用例，一般用覆盖率来衡量测试完整度。

逻辑单位主要有：语句、分支、条件、条件值、条件值组合、路径。语句覆盖就是覆盖所有的语句，其他类推。另外还有一种判定条件覆盖，其实是分支覆盖与条件覆盖的组合。

与条件有关的覆盖有三种：

- 条件覆盖是指覆盖所有的条件表达式，即所有的条件表达式都至少计算一次，不考虑计算结果；
- 条件值覆盖是指覆盖条件的所有可能取值，即每个条件的取真值和取假值都要至少计算一次；
- 条件值组合覆盖是指覆盖所有条件取值的所有可能组合。

一些粗浅的研究发现与条件直接有关的错误主要是逻辑操作符错误，采用分支覆盖与条件覆盖的组合，基本上可以发现这些错误，另一方面，条件值覆盖与条件值组合覆盖往往需要大量的测试用例，因此，条件值覆盖和条件值组合覆盖的效费比偏低。

效费比较高且完整性也足够的测试要求是这样的：完成功能测试，完成语句覆盖、条件覆盖、分支覆盖、路径覆盖。

关于白盒测试用例的设计，程序测试领域的书籍一般都有讲述，普通方法是画出程序的逻辑结构图如程序流程图或控制流图，根据逻辑结构图设计测试用例，这些是纯粹的白盒测试。

也可以先完成黑盒测试，然后统计白盒覆盖率，针对未覆盖的逻辑单位设计测试用例覆盖它，例如，先检查是否有语句未覆盖，有的话设计测试用例覆盖它，然后用同样方法完成条件覆盖、分支覆盖和路径覆盖，这样的话，既检验了黑盒测试的完整性，又避免了重复的工作，用较少的时间成本达到非常高的测试完整性。

3、常见的单元测试框架：

每种开发语言都有很方便的单元测试框架。比如Python语言常用Pytest，java语言常用JUnit，大家可以在网上很方便的查找到自己需要的框架。

适合ATDD/BDD的测试工具有Cucumber、Robot Framework等。

输出物

单元测试用例

单元测试结果

单元测试覆盖率等指标

常见问题解答

开发工程师不愿意或没时间写单元测试怎么办？

1、不是开发的职责？

开发工程师的职责是开发出符合要求的高质量的代码，而单元测试正是保证代码正确的最基本工作，是开发活动的一项。所以单元测试是开发工程师的职责。

2、没时间做？

其实开发过程中的很多操作都已经是单元测试的雏形了，例如写代码前要设计一下分支条件，写完代码要跑一跑，只是把它们作为单元测试用例规范起来就可以了，不会占用很多时间。

执行单元测试速度很快，发现Bug就能立刻修，是效率最高的。越早发现Bug，就能节省更多的时间，降低更多的风险。如果等到提交之后被别人发现Bug再回来改，才是最烦恼的事情，需要花更多精力查找和修改。

只要理念正确了，工程师们都可以做好单元测试的。

还可以学习一下TDD思想和方法哦。

参考资料

1、wikipedia - unit testin

2、百度百科 - 单元测试

3、测试驱动开发之单元测试简介

实战案例

代码开发完成后，进行code review、静态代码检查等，使用Pytest进行单元测试（以Python程序为例）。

在Gitlab上做代码合并时自动执行这些单元测试。测试通过才可以合并到主分支。

冒烟测试

贡献人：Becca



扫码查看详情

定义

冒烟测试（Smoke Testing）在持续集成（Continuous Integration，CI）中被称为BVT（Build Verification Testing，构建验证测试/版本验证测试）。还有被称为版本健康检查（Build Sanity Check）。

冒烟测试是对产品版本的基本功能进行确认验证、以及确定和修复缺陷的最经济有效的方法。

冒烟测试是快速验证产品版本的基本功能是否已经实现，是否会破坏产品版本的稳定性，从而确定能否继续进行更多的测试。

若冒烟测试失败，则不必做进一步的测试，而是直接返回给开发人员。如果通过，再进行后续的功能测试、集成测试、系统测试等等。所以冒烟测试是一个门槛或开关。

基于持续集成（Continuous Integration，CI）的冒烟测试采用自动化测试脚本进行测试工作，提高测试效率，减少测试人员大量的重复测试验证工作。

考虑到持续集成的速度，BVT中只包含关键的功能，如风险高、最重要功能集合、执行时间较短的测试场景。

冒烟测试通过的标准是：冒烟测试用例全部通过，主流程可通过，无阻碍性重大缺陷。

实践出处

冒烟测试这一术语源自硬件行业。对一个硬件或硬件组件进行更改或修复后，先进行通电测试。如果冒烟，说明存在致命的故障，就没有必要做进一步的测试了。如果没有冒烟，则该组件需要继续做进一步的测试。

微软公司把冒烟测试引入了软件领域，即在2000年出版的《微软项目求生法则》一书的第14章提出的一种测试，目的是对一个新编译需要正式测试的软件版本，确认软件的基本功能是正常的，然后才可以进行后续的测试工作。

微软提出冒烟测试，和他一直提倡的每日build（构建版本）有很密切的联系。具体说，冒烟测试就是在每日构建版本建立后，对系统的基本功能进行简单的测试，也用来检验持续集成是否成功。所以冒烟测试也被成为BVT(Build Verification Testing)

为什么

开展冒烟测试工作有助于尽早发现软件代码存在的问题，节约时间节省成本，提高软件代码的质量和开发效率。

冒烟测试，在软件生命周期中所占有的时间比例较低，同时具有注重通过性轻细节的特点，因此经常被开发、测试人员所忽视。事实上，冒烟测试是软件测试过程中一个不可或缺节点，一个好的冒烟测试过程，对于软件测试效率的提升具有重要意义。

- 冒烟测试是对软件质量的总体检验。

通过冒烟测试，能够快速确认软件是否具备测试准入条件，避免出现正式测试阶段全面开展后甚至到测试中后期才发现阻塞型缺陷等严重影响测试进度浪费人力物力的情况。

- 冒烟测试是测试人员对测试流程的熟悉。

通过冒烟测试，测试人员可以迅速熟悉测试总体流程，这一方面有助于测试人员准确制定测试时间计划，合理安排工作进度；另一方面也有助于测试人员提前做好相关设备、数据的准备，为正式测试的开展奠定基础。

何时使用

冒烟测试/BVT，是在所有开发工程师都已经集成完自己的代码，并编译完成，生成当天的版本之后进行，在正式测试前，对产品或系统的一次简单的验证性测试。

- 主要目的是验证最新生成的软件版本功能的完整性，主要的软件特性的正确性，确保全流程没有严重、阻塞性的问题。

如何使用

进行冒烟测试之前需要确定冒烟测试的用例集，对用例集要求覆盖软件的基本功能。这种版本包出包之后的验证方法通常称为软件版本包的门槛用例验证。

冒烟测试用例也称为准入用例，只有准入用例执行通过了，才可以进行新功能的测试。一般来说准入的用例不会太多，大概是全功能用例的十分之一，是测试用例中 PO / level O 的用例。

用例选取遵循以下原则

- 选取业务最重要功能的用例。

重要功能的用例是对软件功能实现具有重要影响的功能模块测试用例，例如：一个数据业务的增删改查功能、一个算法业务的输入输出结果校验等。

- 选取业务最主要流程的用例。

如果在主流程有重大缺陷甚至阻碍，使得业务不可用，那么对分支流程测的再多也无法帮助整体业务的质量。所以冒烟测试要重点关注主要的业务流程。

冒烟测试可以手动执行，也可以自动化执行。冒烟测试的最佳实践还是最好被自动化，在CI中每一个Build都自动的去执行主流程的测试，确保其是一个基本可用的版本。

在持续集成中普遍需要自动化执行。BVT（Build Verification Test）是持续集成的最后一步，用来验证软件的最基本的功能或业务上最常用的功能，也是用于检验持续集成是否成功。

而且，BVT 的测试内容要根据软件版本的演化进行持续更新，增加新的基本特性的覆盖，去掉那些风险较低的测试项。

输出物

迭代的冒烟测试用例集

冒烟测试结果报告

常见问题解答

1、冒烟测试由谁来执行？

- 开发：编码完成后，开发可根据冒烟用例进行自测；
- 产品：验收阶段，产品可根据冒烟用例对产品进行初步验收；
- 测试：根据冒烟用例进行测试验证。

2、冒烟测试 v.s. 回归测试？

- 冒烟测试，是版本验证测试，主要确认新的版本是否存在致命性bug，冒烟测试最大的优点在于节约测试的时间成本，减少测试轮数。
- 回归测试，是软件维护阶段对软件修改后进行的测试，指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。

参考资料

- 1、朱少民老师的敏捷测试书籍和文章，如《敏捷测试 - 以持续测试促进持续交付》
- 2、[百度百科-冒烟测试]

实战案例

使用GitLab持续集成进行每日构建，每天晚上对几个最重要场景进行自动化冒烟测试，用来检验最新版本的基本质量情况。测试结果会实时发到钉钉消息群通知开发人员。

部署和发布

发布策略

发布实施

金丝雀发布

贡献人：徐东伟 晏瑞宇



定义

金丝雀发布就是把应用程序的某个新版本部署到生产环境中的部分服务器中，从而快速得到反馈。就像发现一只煤矿坑道里的金丝雀那样，很快就会发现新版本中存在的问题，而不会影响大多数用户。这是一个能大大减少新版本发布风险的方法。

实践出处

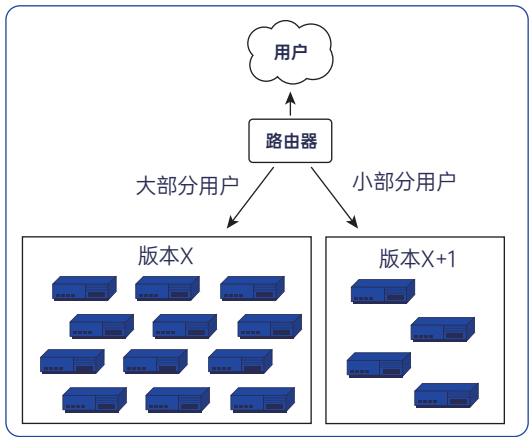
《持续交付》

为什么

金丝雀发布有以下几个好处：

- 非常容易回滚。只要不把用户引到这个有问题的版本上就行了。此时就可以来分析日志，查找问题。
- 还可以将同一批用户引至新版本和旧版本上，从而做A/B测试。某些公司会度量新特性的使用率，如果用的人不多，就会废弃它。另外一些公司会度量该版本产生的收入，如果收入较低，就把该版本回滚。如果软件产生了研究结果，那么可以对新旧版本之间从真正用户那儿得到的结果质量进行对比。你不必使用大量用户对新版本做A/B测试，只要有代表性的样本就足够了。
- 可以通过逐渐增加负载，慢慢地把更多的用户引到新版本，记录并衡量应用程序的响应时间、CPU使用率、I/O、内存使用率以及日志中是否有异常报告这种方式，来检查一下应用程序是否满足容量需求。如果生产环境太大，无法创建一个与实际情况相差不多的容量测试环境，那么这对于容量测试来说，是一个风险相对较低的办法。

如何使用



像蓝绿部署一样，你要先部署新版本到一部分服务器上，而此时用户不会用到这些服务器。然后就在这个新版本上做冒烟测试，如果有必要，还可以做一些容量测试。最后，你再选择一部分用户，把他们引导到这个新版本上。有些公司会首先选择一些“超级用户”来使用这个新版本。甚至可以在生产环境中部署多个版本，根据需要 will 将不同组的用户引导到不同的版本上。

可是，金丝雀发布也并不适用于所有情况。对于那些需要用户安装到其自己环境中的软件来说，这么做就比较困难了。对于这个问题，有另一个解决方案（使用网格计算），那就是让客户软件或桌面应用程序自动从设置的服务器上拿到新版本并自动升级。

金丝雀发布在对数据库升级以及其他共享资源方面引入了更进一步的约束，即任何共享资源（如共享的会话缓存或外部服务等）要能在生产环境中的所有版本中相兼容。另一种方法是使用非共享架构（shared-nothing architecture），即每个结点与其他结点绝对独立，不共享数据库或外部服务，也可以将两种方法结合使用。

最后，在生产环境中保留尽可能少的版本也是非常重要的，最好限制在两个版本之内。支持多个版本是非常痛苦的，所以要将金丝雀发布的数目减少到最低限度。

参考资料

《持续交付》 Jez Humble, David Farley著，桥梁译，人民邮电出版社。

蓝绿部署

贡献人：徐东伟 王增舟 晏瑞宇



扫码查看详情

定义

对于发布管理来说，蓝绿部署是我们所知道的最强大的技术之一。在拥有两套完全相同的生产环境（“蓝环境”和“绿环境”）的前提下，用户正在使用的环境占用其中的一套，将新版软件安装到另外一套环境中，当时机成熟后（如经过生产验证或合适的业务时机），将用户切换到新版本软件所在的环境。

实践出处

由Jez Humble（《持续交付》一书的作者）和Daniel North共同创造。

为什么

部署简单、回滚快速以及灾难恢复轻松，是蓝绿部署的主要优点：

• 测试环境与生产环境相近/相同

通过在与生产系统相同的硬件和软件上执行测试，使测试更加有用和相关。

• 随时部署

我们可以随时发布，因为没有停机时间。无需等待预定的维护窗口。

• 瞬间切换

具有瞬间切换功能，用户几乎可以立即切换到新版本。与此同时，每个人都看到了最新的版本。

• 瞬间回滚

切换可以在双方向上运行。如果我们决定这样做，我们可以立即将所有用户恢复到以前的版本。

• 热备用

蓝绿部署有潜力将我们从灾难中拯救出来。假设其中一个数据中心宕机，从而导致实时环境宕机。这没什么大不了的；我们就换另一个，直到情况解决。只要蓝环境和绿环境的可用性区域不在同一可用性区域中，就会起作用。

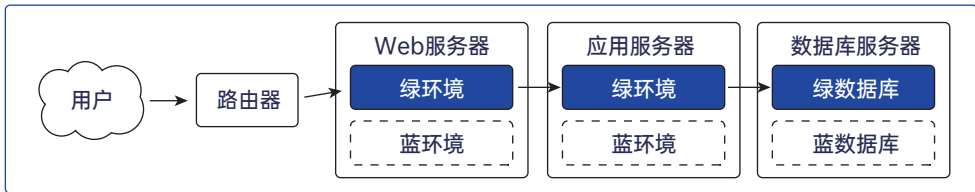
• 方便事后检查

使用就地部署，对失败的发布进行调试就会很困难。因为面临停机时，目标是尽快恢复正常服务。我们无法既要保护现场，又要恢复服务。如果使用蓝绿部署，就可以先将用户切换到能够正常工作的环境版本，我们就可以在有问题的环境中并行进行调试了。

其次，在收集调试数据方面，就地部署方案只能用数据回放的方法来重新构造生产场景，这可能会丢失很多有用的数据。如果使用蓝绿部署就不存在这个问题，因为可以实时接生产数据的真实数据进行并行测试和调试。

如何使用

做法是有两个相同的生产环境版本，一个叫做“蓝环境”，一个叫做“绿环境”。



系统的用户被引导到当前正在作为生产环境的绿环境中。现在我们要发布一个新版本，所以先把这个新版本发布到蓝环境中，然后让应用程序先热身一下(你想多长时间都行)，这根本不会影响绿环境。我们可以在蓝环境上运行冒烟测试，来检查它是否可以正常工作。当一切准备就绪以后，向新版本迁移就非常简单了，只要修改一下路由配置，将用户从绿环境导向蓝环境即可。这样，蓝环境就成了生产环境。这种切换通常在一秒钟之内就能搞定。

如果出了问题，把路由器切回到绿环境上即可。然后在蓝环境中调试，找到问题的原因。

这种方式比重新部署要有一些改进。然而，在做这种蓝绿部署时，要小心管理数据库。通常来说，直接从绿数据库切换到蓝数据库是不可能的，因为如果数据库结构有变化的话，数据迁移要花一定的时间。

解决这个问题的一种方法是在切换之前暂时将应用程序变成只读状态一小段时间。然后把绿数据库复制一份，并恢复到蓝数据库中，执行迁移操作，再把用户切换到蓝系统。如果一切正常，再把应用程序切换到读写方式。如果出了什么问题，只要把它再切回绿数据库就可以了。如果这发生在切成读写方式之前，那么什么额外工作也不需要。如果应用程序中已经写入了一些你想保留的数据，那么，当再次切换回去之前，你就要找到一种方法可以拿到新记录并把它们迁回到绿数据库中。另外，你还可以找个办法让应用程序的新版本把数据库事务同时发向新旧两个数据库。

另一种方法是对应用程序进行一下重新设计，以便能够让迁移数据库与升级流程独立。

如果只有一个生产环境，也可以使用蓝绿部署。只要让应用程序的两份副本一起运行在同一个环境中，每个副本都有自己的资源(自己的端口、在文件系统中有自己的根目录，等等)。这样它们就可以同时运行且互不干扰了。你也可以分别对每个环境进行部署。还有一种方法就是使用虚拟化技术，但是要先测试一下这种虚拟化对应用程序在容量方面的影响有多大。

如果有足够预算的话，蓝绿环境应该是相互完全分离的环境副本。这需要的配置较少，但需要的成本较高。该方法的一种变形[也叫做影子域发布（shadow domain releasing）影子环境发布（shadowen-vironment releasing）或者双热发布(live-livereleasing)]是使用试运行环境和生产环境作为蓝绿环境。将应用程序的新版本部署到试运行环境上，然后把用户从生产环境引导至试运行环境中，让用户开始使用这个新版本。此时，试运行环境就变成了生产环境，生产环境就变成了试运行环境。

参考资料

- 1、《持续交付》Jez Humble，David Farley著，乔梁译，人民邮电出版社。
- 2、<https://www.atatus.com/glossary/blue-green-deployment/>

逐特性发布

贡献人：董越



扫码查看详情

定义

逐特性发布是把持续交付推向极致，每个特性单独进行测试、测试通过后单独进行发布。

实践出处

逐特性发布是去哪儿网至少十年前就全公司使用的一个实践，尽管并没有给它一个正式的名称。

逐特性发布与持续部署有相似的地方。区别是，持续部署里所说的变更，并没有明确的定义。一般认为，持续部署也包括更多的内容，比如全部测试都自动完成、在测试通过后自动发布等。

为什么

持续交付意在使用自动化等一系列手段，在保证质量和安全的前提下，提高部署到类生产环境和生产环境的频率，于是缩短了一个特性从开发完成到发布上线所需的时间。

发布上线的频率越高，一个特性在开发完成后等待凑齐一拨一起发布的时间就越少，一个特性从开发完成到发布上线所需的时间就越短。那么把它推到极致是什么情况呢？一个微服务每天发布1000次？不是。极致是每个特性都单独测试和发布，无需等待其他特性一起测试和发布。没有必要也做不到更高频率了。因为我们对特性的定义就是值得独立发布的最小改动。

何时使用

业务背景越VUCA[Volatility（易变性），Uncertainty（不确定性），Complexity（复杂性），Ambiguity（模糊性）]，越是希望缩短从需求到发布的时间，就越值得考虑逐特性发布的方式。

如何使用

下面介绍逐特性发布方式的典型流程：

- **特性改动累积：**程序员小明把代码改动提交（Push）到服务器端代码库中的特性分支这个动作，自动触发了特性分支上的流水线的运转，进行编译构建、单元测试、代码扫描、生成Docker镜像（如果使用了容器技术）、存入制品库、部署到测试环境、全量的自动化接口测试、基本功能的自动化UI测试。这和逐特性集成时，集成分支上的流水线类似。也有一些区别，比如不是部署到一个“公共”的测试环境，而是部署到每个特性分支专属的测试环境。并不是一定要等到特性开发完成后才去找别人测试。比如这次，在完成了一部分功能后，小明就让产品经理来看一下，让测试人员测一下。他们随时面对面交流。另外，如果有必要的话，还会做性能测试、安全测试等非功能测试，这些测试基本是自动化的。

- **特性改动发布：**当确定都没问题后，小明就会在流水线上点个按钮，检验确定当前版本是基于最新已发布版本，然后把这个新版本部署到生产环境中的一台机器或一个容器上，随后自动引入1%的流量，看看监控指标是否正常，以及那些用户会不会疯狂地给客服人员打电话……如果没有出现这种情况，那么就会自动的继续分批部署到更多机器或容器并引流，直到100%的流量都被引到新版本上，发布完成。随后是善后工作，自动将特性分支合并到master分支，因为master分支代表着最新已发布版本。

下面介绍逐特性发布的实现要点：

- 测试环境自动创建与分配。由于每条特性分支都需要相应的一个特性测试环境，流水线也需要它，人工测试也需要它。因此特性测试环境的数量是不固定的，随时可能需要更多的特性测试环境。当需要一套新的特性测试环境时，最好是可以很快很方便地创建出来。这就要求创建过程是自动化的，而且不需要繁琐的审批流程。当想要的时候，应该就是点个按钮的事儿。更进一步，考虑到创建毕竟还是需要一些时间，最好是总是保持少量已创建好的且空闲的测试环境实例，随时可以分配。

- 必要时使用虚拟独占方式。当整体环境中包含了成百上千个微服务，那么提供一套测试环境就需要很高的成本，于是仅能有一两套功能测试环境，仅在特性分支合入集成分支后才做正式测试。此时，在技术上就限制了流程的优化。解决的方法是，一套测试环境，也就是一个整体环境实例，并不意味着其中每个微服务都是这个环境实例独占的——一个微服务实例可以为多个环境实例服务，而在技术上实现让一个微服务实例仿佛只为一个环境实例服务，不同环境实例之间看起来相互隔离。我们姑且称这一类方法为虚拟独占方式。这需要借助一些技术和方法来实现，比如通过消息路由、远程调用路由等来实现，这与具体系统使用的微服务架构有关，不能一概而论。

- 特性分支上持续测试。在特性分支上，不仅是代码提交要触发流水线进行一系列自动化的检查，人工测试也应当尽量在开发过程中就做，而不是等到这个特性开发完成后。在特性开发完成后再做，测试以及随后的修复，就会变成关键路径上明显的一段。

- 消除发布窗口。发布窗口是指只有在特定的日期和/或特定的时间才能发布。发布窗口造成积压和等待，使得逐特性发布变得没有意义，因为反正要等到发布窗口才能发布。

- 消除发布审批流程。发布不太频繁时，有个发布审批流程，尚能接受。当发布很频繁，每个特性都单独发布时，发布审批流程会让开发团队和领导都不胜其烦。反过来，考虑到发布审批流程会让开发团队和领导都不胜其烦，大家就不想频繁地发布。

- 开发团队自行发布。在测试通过后，开发人员或测试人员点个按钮，就发布了。发布不太频繁时，转给运维人员操作发布，尚能接受。当发布很频繁，每个特性都单独发布时，这道转手会让开发团队和运维人员都不胜其烦。反过来，考虑到这道转手会让开发团队和运维人员都不胜其烦，大家就不想频繁地发布。

灰度发布

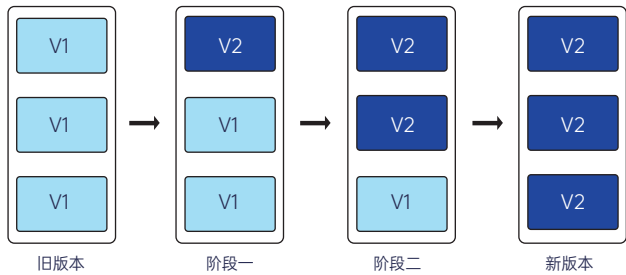
贡献人：何凡



扫码查看详情

定义

灰度发布是金丝雀发布的一个扩展，它把发布分为几个阶段，每一阶段的用户数量逐步递增。如果在当前阶段，新版本没有发现问题，那么将会在下一阶段中添加更多的用户，直到所有的用户都能使用。



为什么

灰度发布是通过在并存版本间切换路由权重的方式来达到不同版本逐步切换的目的，是一种无停机的发布策略，这种策略可以降低发布风险，对用户基本上没有影响。

何时使用

在如下情况可以考虑使用灰度发布策略：

- 1、对可用性要求高（即不停机，对用户影响小）。
- 2、对发布风险控制要求高。

如何使用

灰度发布通过在并存版本间切换路由权重，使不同版本逐步切换。

需要注意的是，整个发布周期会比较长，新旧版本并存，因此在开发的时候，要考虑到不同版本的兼容性，新旧版本并存不能对功能的使用带来任何影响。在新版本出问题的时候，灰度发布可以迅速地回滚到旧版本。

可以配合使用特性开关等技术，可以使灰度发布策略更加灵活。

常见问题解答

灰度发布有哪些优缺点？

- 优点：
 - 随时能回滚到上一版本，发布风险降低；
 - 无需停机发布，对用户基本上没有影响。
- 缺点：
 - 要考虑新旧版本兼容性，对版本设计要求高；
 - 发布系统复杂，对运维要求高；
 - 发布周期比较长，对版本上线时间紧张的任务不友好。

参考资料

- 1、《阿里巴巴DevOps实践指南》
- 2、<https://developer.aliyun.com/topic/devops>

部署人员参与设计部署流程

贡献人：徐东伟



扫码查看详情

定义

部署人员在项目启动时就参与其中，了解项目，为之后部署的执行奠定基础。

实践出处

《持续交付》

为什么

部署人员和研发人员都可以非常清楚要发布的东西以及相关的注意事项，让发布能够非常顺滑进行。

何时使用

从项目启动开始，部署人员就可以适时参与。

如何使用

在一个项目启动时，研发人员首先要做的一件事情就是非正式地找到部署人员，让他们也参与到开发过程中。这样，部署人员从项目一开始就已经参与了软件开发，双方都知道在发布之前发生了什么，因此在部署的时候一切都变得非常丝滑。

参考资料

《持续交付》 Jez Humble，David Farley著，桥梁译，人民邮电出版社。

运维

基础设施服务

人员配备

创建共享服务

贡献人：徐东伟



扫码查看详情

定义

创建共享服务，就是创建一套集中式的平台和工具集服务，让所有开发团队都能够通过使用这套平台和服务来提高生产力，例如搭建类生产环境、部署流水线、自动化测试工具、生产环境遥测控制台等。

在理想情况下，运维提供的所有平台和服务应该是全自动化的，并且能按需提供，不需要发人员提交工单，然后再等待运维团队手动处理，这能保证运维不成为客户的瓶颈(例如“我们已收到您的工单请求，手动配置这些测试环境需要 6周”)。

实践出处

《DevOps实践指南》

为什么

通过使用共享服务，开发团队就能够把更多的精力和时间用在功能构建上，而不是消耗在获取交付和支持生产环境特性所必需的基础设施上。这种方式使得产品团队能够及时地取得所需的资源，同时降低了沟通和协作成本。

何时使用

时刻准备着，按需提供实时服务。

如何使用

大多数情况下，我们都不会向内部团队强推这些平台和服务—这些平台团队必须通过令人满意的服务来赢得内部客户，有时甚至需要同外部供应商竞争。通过在内部建立这种有效的市场竞争机制，可以确保所提供的平台和服务是最易于使用的，而且是值得选择的。

例如，可以搭建一个这样的平台：内置可共享的版本控制系统，其中包含可安全使用的库；提供能自动执行代码质量和安全扫描工具的部署流水线，还能把应用程序部署到装有遥测工具的生产环境里。在理想情况下，通过使用这种平台，开发团队的工作会变得更轻松，他们会认为使用这个平台是最简单、最安全、最可靠的工作方式。

通过汇聚包括QA人员、运维人员和信息安全人员在内的全体人员的集体经验，我们能够创建更安全的工作系统。这不仅能提高开发人员的生产力，还让产品团队可以轻松地使用工具和流程，例如执行自动化测试，以及满足信息安全性和合规性要求。

创建和维护这些平台和工具是真正的产品开发——客户不是外部客户，而是内部开发团队。像构建任何一个伟大的产品一样，做出一个人见人爱的平台绝非偶然。如果对客户的关注度不够，则内部平台团队有可能开发出令人讨厌的工具，导致客户迅速放弃它，并去寻找新的替代品，甚至转向另一个内部平台团队，或者联系外部的供应商。

Netflix工程工具总监Dianne Marsh称，她的团队宗旨是：“以支持工程团队的创新和速度为先。我们不为这些团队构建、打包或部署任何产品，也不为他们管理配置。相反，我们构建自助工具。他们可以依赖我们的工具，但不能依赖我们的劳动力。”

平台团队通常还能提供其他服务，帮助客户学习他们的技术，或者做技术转移，甚至还提供指导和咨询，从而提升组织内部的实践水平。他们提供的共享服务有利于促进标准化，即使他们在不同团队之间切换，也能让工程师快速进入状态。例如，如果每个产品团队都选择不同的工链，则工程师不得不学习一套全新的技术来完成工作，这样就使团队目标优先于全局目标了。

在有些组织中，团队只能使用那些经过批准的工具。在这种情况下，可以尝试对少数团队（例如DevOps转型团队）解除这些限制，从而通过实验发现可以提高团队效率的工具。

内部共享服务团队应该不断地发掘能在组织内部广泛应用的工具链，判断哪些是能通过集中式平台提供的，并让每个人都可以使用。一般来说，发现经实践验证的工具并拓展其使用范围，要比从零开始构建这些功能更容易成功。

输出物

共享服务

参考资料

《DevOps实践指南》 【美】 Gene Kim, Jez Humble, Patrick Debois, John Wills著，刘征，王磊，马博文，曾朝京译，中国工信出版集团，人民邮电出版社。

为每个开发团队分派运维联络人

贡献人：徐东伟



扫码查看详情

定义

给每个开发团队指定一位运维联络人。

实践出处

《DevOps实践指南》

为什么

通过为每个开发团队分派运维联络人，目标是确保运维不会成为开发团队的瓶颈。

何时使用

这是在由于各种原因（如成本或资源不足），无法实施“运维工程师融入开发团队”实践的情况下可以采取的策略。

如何使用

集中式运维团队依然管理着所有环境（不只是生产环境，还包括预生产环境），负责确保它们的一致性。派遣到开发团队的运维工程师的责任是理解下列内容：

- 新产品的功能是什么，为什么要开发这个产品；
- 它是怎样工作的，可运维性如何，可扩展性和监控能力如何（强烈建议以图示说明）；
- 怎样监控和收集指标，如何确认应用的功能正常；
- 架构和模式是否与以往的做法不同，这样做的理由是什么；
- 是否对基础设施有额外的需求，它的使用对基础设施容量的影响如何；
- 特性的发布计划。

此外，与融入运维工程师的模式相同，运维联络人也要参加开发团队的站会，把开发团队的需求纳入整体的运维计划，并且在必要的时候执行相关任务。在发生资源竞争或优先级冲突时，团队依赖运维联络人推进问题的解决。通过这种方式，我们可以在更宽泛的组织目标背景下，评估和处理资源竞争和冲突。

相比融入运维工程师的模式而言，分派运维联络人的方式能支持更多的开发团队。我们的目标是确保运维不会成为开发团队的瓶颈。如果发现由于运维联络人的工作量过大而导致开发团队无法实现目标，那么就可能需要减少每个联络人所支持的团队数量或者临时将运维工程师融入某些开发团队。

参考资料

《DevOps实践指南》【美】Gene Kim, Jez Humble, Patrick Debois, John Wills著，刘征，王磊，马博文，曾朝京译，中国工信出版集团，人民邮电出版社。

运维工程师融入开发团队

贡献人：徐东伟



扫码查看详情

定义

运维工程师融入开发团队，运维工程师的工作优先级几乎完全受所在研发团队的目标驱动。

实践出处

《DevOps实践指南》

为什么

运维工程师融入开发团队，使开发团队能自给自足，降低对集中式运维的依赖程，研发团队可以完全负责服务的交付和支持。

这种范式有一个重要优势：开发团队和运维工程师的紧密配合和协作是一种极其有效的方式，能将运维知识通过交叉培训的方式融入服务团队，还可以将运维知识逐渐转换为自动化的代码，使之更可靠和更广泛地重用。

何时使用

如果有充足的运维工程师和充足的预算，将运维工程师融入开发团队是优选方案。

如何使用

开发团队通常要准备专用的预算雇用这些运维工程师，不过面试和聘用决策可能还是由集中式运维团队来完成，以确保一致性和员工的素质。

迪士尼的系统工程总监Jason Cox说：“迪士尼将运维人员（系统工程师）融入业务部门的产品团队、开发团队、测试团队，甚至信息安全团队。这完全改变了我们的工作方式。运维工程师创建工具并转变他人的工作方式甚至想法。在传统的运维模式里，我们只是驾驶别人建造的列车：而在现代化的运维模式里，我们不仅协助建造，而且还帮助列车安全地行驶。”

对于新的大型开发项目，可以在启动阶段就融入运维工程师。他们的工作包括参与做什么和如何做的辅助决策，影响产品架构，辅助内部和外部的技术选型，帮助创建内部平台的新功能，甚至产生新的运维能力。当产品上线之后，运维工程师可以帮助开发团队承担运维责任。

他们将参加开发团队的相关讨论，如计划会议、每日站会以及新特性的演示，并帮助决定可以交付哪些特性。

随着开发团队对运维知识和能力的需求逐渐降低，运维工程师就可以转移到其他的项目或者工作中，接着按照以上模式进入下一个团队以及相应产品的生命周期。

参考资料

《DevOps实践指南》【美】Gene Kim, Jez Humble, Patrick Debois, John Wills著，刘征，王磊，马博文，曾朝京译，中国工信出版集团，人民邮电出版社。

度量

交付价值

交付速率

交付质量

交付成本

交付能力

持续改进

综合度量

客户问题解决时长

主要贡献人：晏瑞宇



扫码查看详情

定义

客户问题解决时长是客户问题解决时间与客户问题首次响应时间之差。指标反映了解决客户问题的及时性，是 SLA 的关键指标，也是客户体验的关键指标之一。

如何使用

计算方法：客户问题解决时长 = 客户问题解决时间-客户问题首次响应时间；
客户问题解决时间和客户问题首次响应时间应根据企业业务实际情况和客户服务方式自行定义。

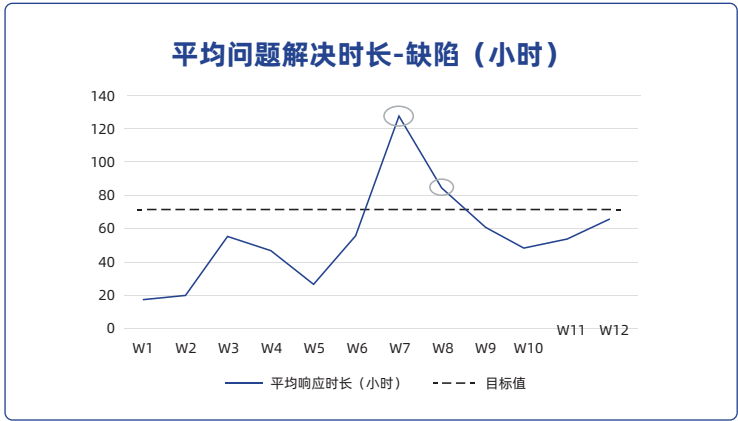
计量单位：时长；

涉及阶段：需求、测试、发布、运营；

补充说明：客户问题解决时长，可进一步按关注的维度进行统计，并和目标值进行对比，以利于及时进行分析和采取纠正措施。客户问题解决时长的实际值与目标值的对比分析，有利于促成快速响应的针对性改进。

常用的关注维度包括：

- 时间维度，如：天、周、月等；
- 问题维度，如：按需求、缺陷、变更等；
- 级别维度，如：高、中、低等级；
- 组织维度，如：项目、产品线、部门等。



客户问题首次响应的时间和客户问题的解决时间应关联相应的关注维度，宜以 IT 系统支持。
记录客户问题首次响应时间是一个关键数据，通常由于内部流转次数过多而导致反复沟通会对客户体验产生较大影响。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

客户问题响应时长

主要贡献人：晏瑞宇



扫码查看详情

定义

客户问题响应时长是客户问题响应时间与客户问题提出时间之差。指标反映了响应客户的问题的及时性，是 SLA 的关键指标，也是客户体验的关键指标之一。

如何使用

计算方法：客户问题响应时长 = 客户问题响应时间-客户问题提出时间；
其中，客户问题响应时间和客户问题提出时间应根据企业业务实际情况和客户服务方式自行定义。

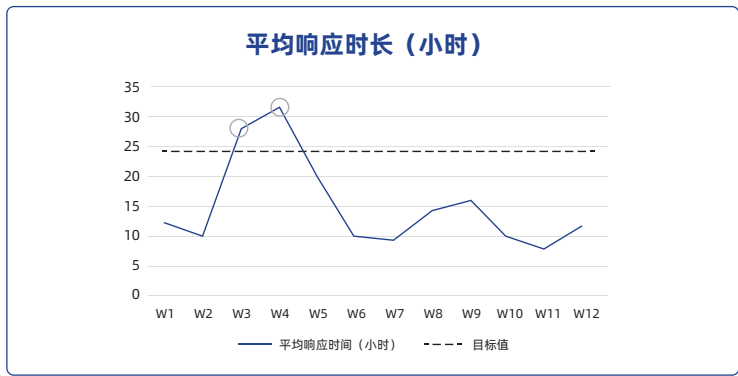
计量单位：时长；

涉及阶段：需求、测试、发布、运营；

补充说明：客户问题响应时长，宜进一步按关注的维度进行统计，并和目标值进行对比，以利于及时进行分析和采取纠正措施。客户问题响应时长的实际值与目标值或标准响应时长的对比分析，有利于促成快速响应的针对性改进。

常用的关注维度包括：

- 时间维度，如：天、周、月等；
- 问题维度，如：按需求、缺陷、变更等；
- 级别维度，如：高、中、低等级；
- 组织维度，如：项目、产品线、部门等。



客户问题提出时间和客户问题的响应时间应关联相应的关注维度，宜以 IT 系统支持。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

客户满意率

贡献人：晏瑞宇



定义

客户满意率是产品交付满足客户/用户需要和期望的程度。指标反映了客户/用户综合体验，通常是关键业务指标之一。

如何使用

计算方法：客户满意率 = $\sum(\text{分项得分权重})/(\text{分项总分权重})$ ；

客户满意度分值，通常包括很满意、满意、基本满意、不满意，亦可按对应取值或取值范围进行打分，如下表所示：

定序取值	对应取值	取值范围
很满意	100	90-99
满意	80	80-89
基本满意	60	60-79
不满意	0	0-59

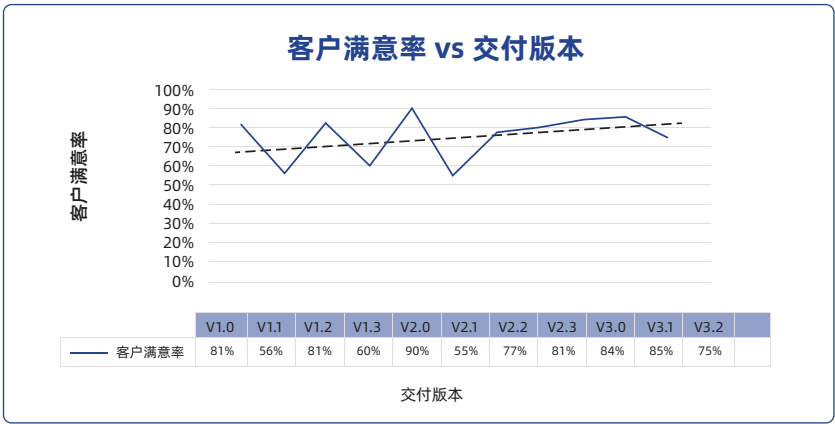
客户满意度调查，应设定几个分项，针对各分项应设定相应权重。分项的设立可以按认知域定义，也可以用需求实现、产品质量、进度控制、服务能力、管控能力等更通俗的用词定义。

计量单位：%；

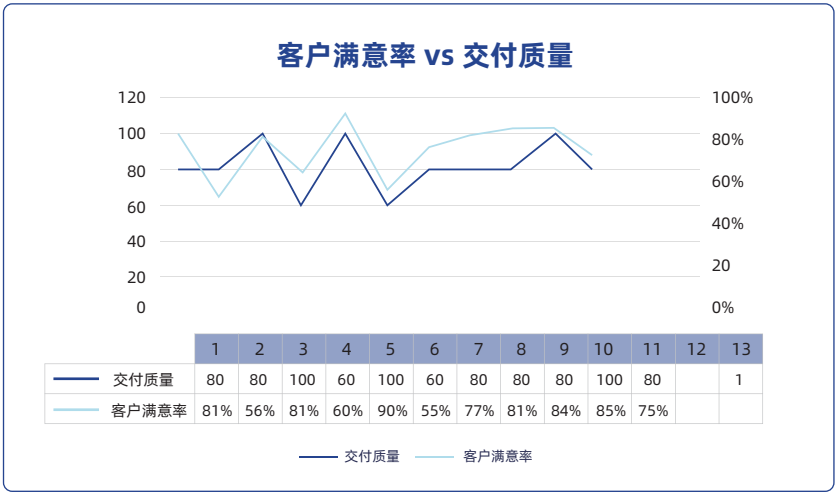
涉及阶段：发布、运营；

补充说明：为了充分利用客户满意度数据，应收集必要的属性，如所属：部门、产品线、产品、项目、版本、满意度调查时间等，以便进行分析和改进，示例如下：

示例1：客户满意率 vs. 交付版本，展现了随着各个版本的交付，客户满意度调查的结果的变化。



示例2：客户满意率vs.交付质量得分。使用时序分析，针对总分（客户满意率）和分项（如：交付质量得分）进行比较，显示交付质量得分的变化对客户满意率的影响。



在客户满意度调查中，应遵循及时有效原则，如：在发布后及时推送并获取客户满意度反馈。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

市场占有率

贡献人：晏瑞宇



扫码查看详情

定义

市场占有率是统计范围内企业销售或运行的产品台套数占市场整体的比例。指标反映了企业在市场上的产品竞争力和销售能力。

对于特定行业，市场占有率会有可靠的数据来源。在产品生命周期的不同阶段，由于采用策略不同，市场占有率也会呈现不同的比例。

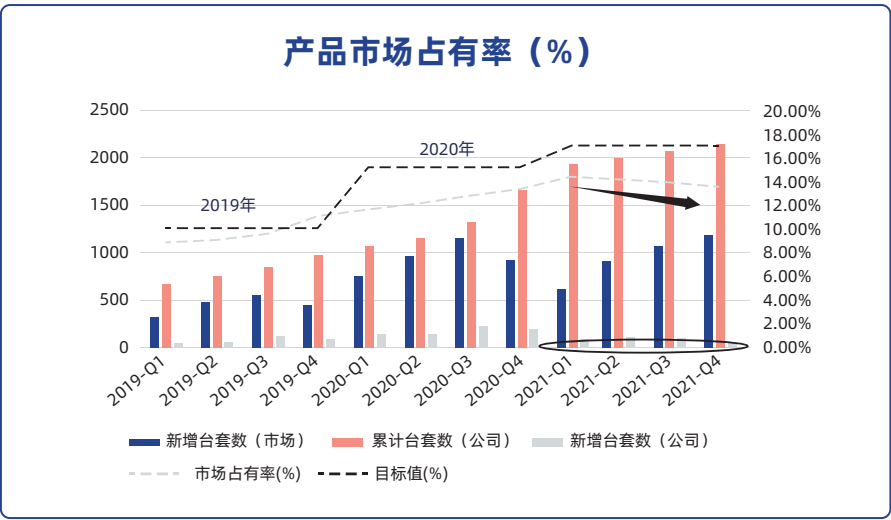
如何使用

计算方法：市场占有率 = 销售或运行的企业产品台套数/市场上销售或运行的同类产品台套总数；

计量单位： %；

涉及阶段： 运营；

补充说明： 基于市场统计数据，以月度/季度等为统计范围，展现市场上企业产品累计台套数和市场占有率，并和每年的目标值进行对比。宜关注公司产品新增台套数与市场新增台套数的比例、增长趋势，识别异常原因，以支持决策和专项改进，示例如下：



参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

用户增长率

贡献人：晏瑞宇



扫码查看详情

定义

用户增长率是统计范围内增加的用户数量相对于用户总量的比例。指标反映了企业的产品能力和运营能力。

对于特定行业，用户增长率是营收增长的先行指标，可按业务线、产品线进行统计，并与营收增加额进行关联分析。

如何使用

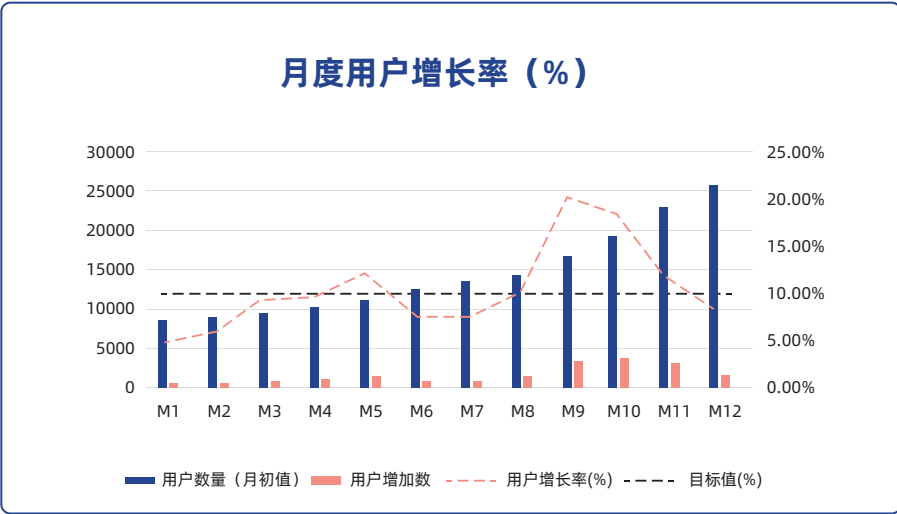
计算方法：用户增长率 = 增加的用户数/用户总数；

增加的用户数可能是负数，以反映用户数量的减少；

计量单位： %；

涉及阶段： 运营；

补充说明： 按照设定的统计范围，统计增加的用户数量（+/-），计算用户增长率，并和目标值进行对比，分析用户增长趋势，并识别异常原因，有利于支持决策和专项改进，示例如下：



进一步的数据分析，可进行环比和同比分析，并考虑周期性模式（如：季节性/节假日）的影响，以建立对数据的深入理解，并支持业务决策。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

营收增长率

贡献人：郑乔尹



扫码查看详情

定义

营收增长率是统计范围内增加的营收相对于总营收的比例。指标反映了企业销售和经营的能力。

通常按业务线、产品线统计营收增长率，并进行同比和环比分析，以评判签约、回款和利润变化，支持决策分析。

如何使用

计算方法： 营收增长率 = 增加营收/总营收；

计量单位： %；

涉及阶段： 运营；

补充说明： 在实际统计分析中，应注意主营业务和增长业务数据的分析和判定，通常可扩展为签约、回款、利润等。尽管有的企业按季度进行考核，也需按月度进行监控，并注意内在模式的影响，如：一季度通常会受春节和规划影响、三季度往往是高峰、四季度往往回落等。部门经理、产品经理应关注业务和产品价值的财务数据，并与研发效能度量及提升相结合。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

产品价值达成率

贡献人：郑乔尹



扫码查看详情

定义

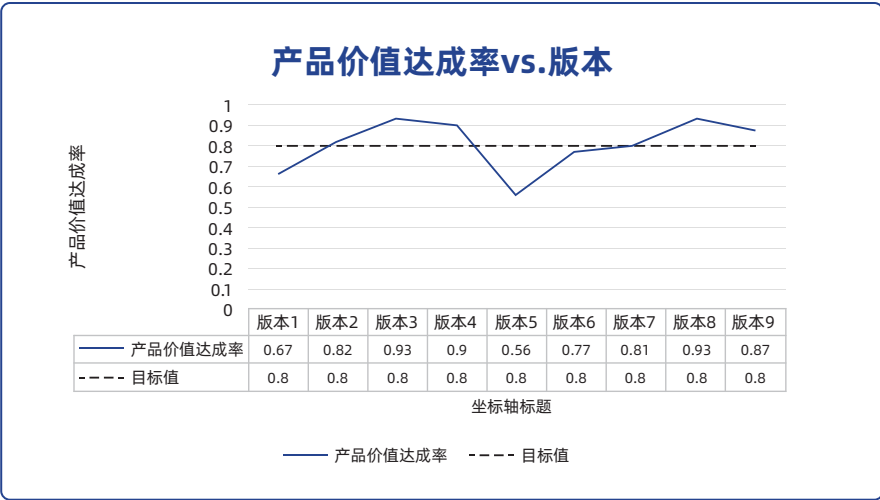
产品价值达成率是产品对用户需求满足的程度。指标反映了是否在用户规定的时间内完整、正确地完成了用户提出的需求。

产品价值达成率是由客户或业务代表进行打分的方法进行计算得出，应在策划/需求澄清环节和产品交付/演示环节分别评分。

为什么

随着多版本的持续交付，产品价值达成率指标体现了团队交付的产品或服务的时序变化。

如果与目标值/基线值比较，则能突出产品价值的变化，利于业务决策。



产品价值达成率，可以按项目、产品、产品线、部门进行统计和分析，目标值设定可根据实际业务需要设定，其变化趋势有助于支持决策和改进。

如何使用

计算方法：

产品价值达成率 = $\sum$ （产品价值评分权重） / $\sum$ （需求价值评分权重）；

其中需求价值评分通常在策划/需求澄清环节，产品价值评分通常在产品交付/演示环节，分别由客户或业务打分。

需求概述	策划/澄清 (1-10)	交付/演示 (1-10)	权重
需求 1	8	5	10.0
需求 2	9	10	3.0
...	...	...	...
需求 N	10	6	1.0
加权分值	117	86	
达成率		73.5%	

计量单位：%；
涉及阶段：需求、发布。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

需求颗粒度

贡献人：晏瑞宇



扫码查看详情

定义

需求颗粒度即单个需求的大小或规模。指标反映了需求进行设计开发的复杂度。通过需求颗粒度的估算和核算，逐步提高对需求粒度的把握，有利于提高计划可行性、资源利用率、交付速率等。

如何使用

计算方法：需求颗粒度 = 需求的大小或规模；

可提取各个需求的复杂度相关的属性，如故事点、功能点对应代码提交的当量；

计量单位：功能点/故事点；

涉及阶段：需求；

补充说明：需求颗粒度可根据企业业务实际情况采用故事点、功能点等进行估算。在需求开发之后，需求颗粒度可以通过其实际数据进行核算，例如采用对应代码的行数、代码当量等。通过估算和核算分析差异，逐步校准团队对需求颗粒度的把握，有利于项目管理中事务的合理安排和顺畅流转，提高资源利用率。

下图给出了故事点数和代码当量对比分析的例子，如果故事点数与代码当量的折算率过高或过低，应进行讨论，提升团队对需求颗粒度的预估和把控。

需求编号	故事点数	代码当量
MESS-9917	8	1173
MESS-8456	8	643
MESS-9608	5	1992
MESS-6765	5	1149
MESS-9807	5	793
MESS-9812	5	494
MESS-9820	5	232
MESS-7099	5	182
MESS-9110	3	803
MESS-5726	3	398
MESS-9733	3	356
MESS-9086	3	315
MESS-7124	3	280
MESS-9640	3	246
MESS-9104	3	232

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

代码提交频率

贡献人：晏瑞宇 徐东伟



扫码查看详情

定义

代码提交频率是在单位时间内，代码版本控制系统记录的代码变更次数。指标反映了代码开发的活跃度，同时可用于提倡小步提交。

可按项目、团队（个人）、版本等维度统计提交频率，展示一定时间段（如最近一周或一天）的代码开发活动。

如何使用

计算方法：代码提交频率 = 代码提交次数 / 时长；

计量单位：次/时长；

涉及阶段：开发；

补充说明：代码变更宜小步提交，有利于代码评审，并尽早将变更暴露给其他协作的开发者，降低代码合并和缺陷修复成本。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

接口变更率

贡献人：晏瑞宇



扫码查看详情

定义

接口变更率是在统计范围内，接口确认后发生变更的比例。指标反映了接口设计及变更控制能力。约定的接口如果发生变更，通常会对其他组件产生影响，并带来额外的成本或返工。

如何使用

计算方法：接口变更率 = 变更的接口数量/接口总数；

计量单位：%；

涉及阶段：需求、设计、开发、测试；

补充说明：

1、接口变更率可按照变更发生的阶段进行分类统计，以利于分析和控制。

2、接口定义、确认和变更在开发流程中应有完整的记录。有的代码分析工具可用于统计接口变更率，包括：需要指明接口文件或函数；统计发生了增删改并通过测试的接口并可作为接口总数；进而统计这些接口定义的代码发生的变更，视为接口定义的变更并计算变更的接口数量。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

组件复用率

贡献人：晏瑞宇



扫码查看详情

定义

组件复用率是在统计范围内复用组件的数量占组件总数的比例。指标反映了组件的功能、性能、质量和可扩展性，也一定程度上反映了设计能力。组件复用率的提升，将显著提升交付速率和质量。

如何使用

计算方法：组件复用率 = 复用的组件数 / 组件总数；

计量单位：%；

涉及阶段：设计、开发；

补充说明：在实际应用中，可进一步定义组件复用度（%），即组件复用程度的百分比，以增加可操作性，支持持续改进。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

组件按时交付率

贡献人：晏瑞宇



扫码查看详情

定义

组件按时交付率是在统计范围内，组件按时交付的数量占交付组件总数的比例。指标反映了组件按期交付能力，也一定程度反映了计划能力。

如何使用

计算方法：组件按时交付率 = 按时交付的组件数 / 计划交付组件数；

计量单位：%；

涉及阶段：设计、开发、测试；

补充说明：在组件化开发模式下，组件的按期交付对产品集成、测试和发布有显著影响。当基于组件和平台进行定制化开发时，延期交付的组件，可能会迫使定制开发团队重复开发，造成资源浪费、质量低下和维护困难。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

需求变更率

贡献人：郑乔尹



扫码查看详情

定义

需求变更率是在统计范围内，需求确认后发生变更的数量占已确认需求总数的比例。指标反映了需求理解及变更控制能力，经确认后发生的需求变更尽管难以完全避免，但其变更对项目进度、质量、成本等往往带来不利影响。

如何使用

计算方法：

需求变更及其次数可从项目管理软件的记录中获得；
需求变更率 = 发生过变更的需求数 / 已确认需求数；

计量单位： %；

涉及阶段： 需求、设计、开发、测试、发布、运营；

补充说明： 需求变更率应首先基于客户视角进行统计，计算需求变化的比例，以利于沟通和协调。进一步的统计分析可考虑需求颗粒度（如功能点、故事点等）和变更工作量的影响，以利于对标和决策。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

需求按时交付率

贡献人：郑乔尹



扫码查看详情

定义

需求按时交付率是在统计范围内需求按时交付的数量占交付需求总数的比例。指标反映了需求按期交付能力，也一定程度反映了计划能力。

如何使用

计算方法： 需求按时交付率 = 按时交付的需求数 / 计划交付需求数；

计量单位： %；

涉及阶段： 需求、交付。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

里程碑偏差

贡献人：郑乔尹



扫码查看详情

定义

里程碑偏差是指在统计范围内，里程碑实际完成时间与计划完成时间之差。指标反映了项目进度，以及里程碑的计划和控制能力。

如何使用

计算方法：里程碑偏差 = 里程碑实际完成时间 - 里程碑计划完成时间；

计量单位：天；

涉及阶段：开发、测试、发布；

补充说明：亦可使用里程碑偏差率（里程碑偏差/计划周期）反映偏差的比例。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

开发交付周期

贡献人：郑乔尹 liuyue



扫码查看详情

定义

开发交付周期是需求被研发团队确认，到完成开发、测试、业务验收，达到可上线状态的时长。指标反映了研发团队的交付速率，依赖需求拆分和版本规划以及研发、测试和业务团队的分工协作。

如何使用

计算方法：

开发交付周期 = 需求验收时间 - 需求确认时间；
需求验收时间是指通过业务验收，达到可发布状态的时间，需求确认时间是指研发确认需求并纳入版本计划的时间。

计量单位：天；

涉及阶段：开发、测试、发布；

补充说明：软件发布通常包括内部发布和外部发布。基于版本规划，根据市场或客户需要选择适宜的发布版本对市场或客户发布，也是建立业务弹性和降低风险能力的体现。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

需求交付周期

贡献人：郑乔尹 徐东伟 晏瑞宇 liuyue



扫码查看详情

定义

需求交付周期是从需求创建到发布的时长。指标反映了整个团队（包括业务、产品、设计、开发、测试、运营等各职能）针对客户问题或业务机会的交付速率，也是团队协作能力的体现。

可按照需求状态统计软件生命周期各阶段的时间占比，以支持分析和决策。

如何使用

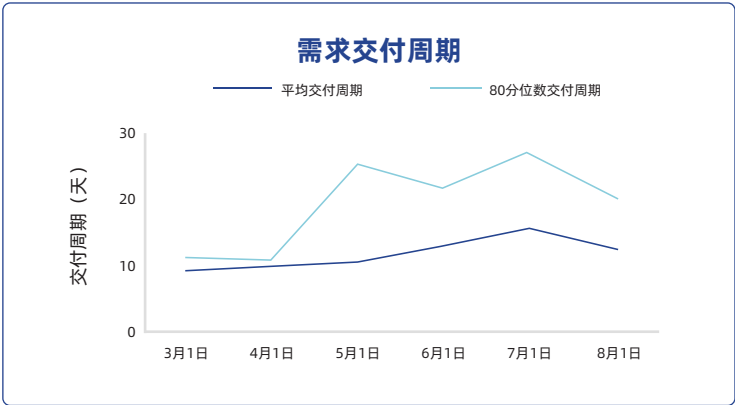
计算方法：

需求交付周期 = 需求发布时间 - 需求创建时间；

可按照需求在各个状态中停留的时长，将需求交付周期按照软件生命周期各个阶段细分为多个组成部分。企业可根据业务实际情况定义需求交付是指单个需求还是批次需求，也可使用业务需求评审或确认时间代替需求创建时间，以便获取明确的时间节点。

计量单位：天；

补充说明：下图展示了几个月份中，需求交付周期的均值和 80%分位值。



需求交付周期反映研发团队的快速响应能力。通常，向客户交付价值的速率越快越好，但必须综合考虑交付价值是否满足顾客期望、需求吞吐量和交付质量等其他方面。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

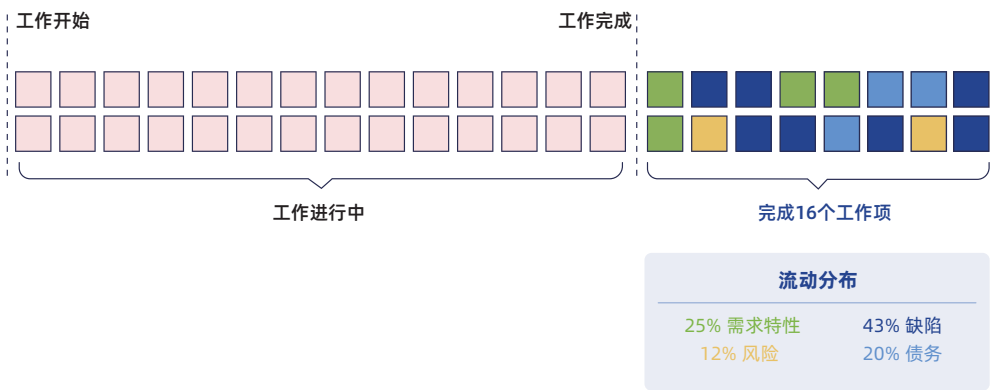
流动分布

贡献人：张乐 晏瑞宇



扫码查看详情

定义



流动分布通过显示在给定时间内完成的流动项（特性、缺陷、风险和债务）的比例，来衡量在不同价值创造类别中的实际投入。

实践出处

《Project to Product》一书

为什么

利用流动分布来为价值流中不同类型的工作带来可见性，这样就可以从优先级的角度看到当前投入的重点在哪里。

何时使用

建议作为日常观察指标。

如何使用

如果统计出来的流动分布（相当于资源的分配）与业务优先级不一致，则需要进行调整。流动分布通过使资源的分配可见，推动产研团队与业务需求方进行各类工作优先级的权衡，而这里的权衡是一种零和游戏。另外，流动分布随着时间的推移，根据产品所处的阶段，需要进行持续调整和演化。

指标解读

- 缺陷占比很高：一般表明缺陷和未计划的工作降低了需求交付的能力，对于技术债务的投资可能需要加强。
- 缺少技术债务和风险的占比：相关工作被忽视或延后，虽然短期看起来交付的需求多了，但未来可能会出现债务危机。

常见问题解答

每条价值流的流动分布应该如何设置？

流动分布会随着时间的推移，不断发生演化。新产品的价值流通常被调整为最大化需求交付的比例。一旦该产品上市并且有稳定用户，就有必要构建额外的能力来处理可能出现的支持工单和故障，并且还要安排一些工作来减少在随后发布周期中不断积累的技术债务。

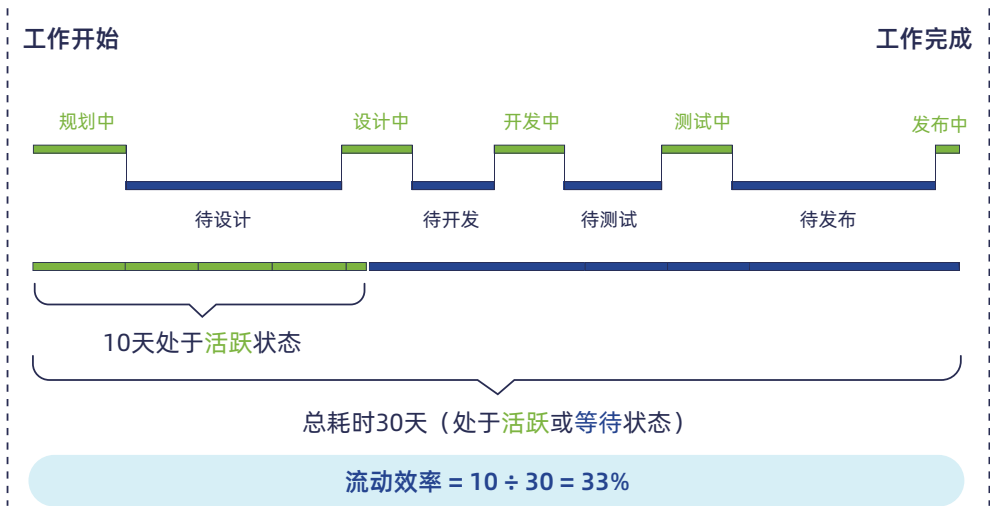
流动效率



扫码查看详情

贡献人：张乐 晏瑞宇 徐东伟

定义



流动效率指流动项处于活跃工作状态的时间占总消耗时间的比例。

实践出处

《Project to Product》一书

为什么

度量流动效率，可以帮助团队从瓶颈中可视化等待时间，以便找出导致流动停滞的问题。流动效率越低，工作停滞在等待状态的时间就越长。

何时使用

建议作为日常观察指标。

如何使用

此指标可以与其他流动指标结合使用，重点聚焦于减少等待时间。

指标解读

- 数值较高：流动效率越高，一般表明交付过程越顺畅、越没有阻塞。但也要警惕指标过高的情况（例如超过40%），这可能意味着状态映射错误或不准确，比如把实际上是等待的阶段映射成活跃状态了，这样就会导致这个指标虚高。

- 数值较低：一般表明存在瓶颈、低效的流程、过多的依赖关系、资源匮乏等，这些问题会导致流动负载的增加，更长的队列，以及更长的流动时间。

常见问题解答

1、流动效率基于流动时间还是周期时间来计算？

流动效率是基于流动时间而不是周期时间，它涵盖了开发上下游的等待时间。如果开发团队在等待用户界面设计，而设计人员被分配到了其他工作，则流动效率会下降，因为相关需求处于等待状态，原因是这两个团队都没有处理它们。因此，可以通过追踪流动效率降低的原因来识别价值流的瓶颈。

2、流动效率的正常值应该多少合适？

我在行业中经常看到一些统计数据，很多企业的实际流动效率要比想象中低很多，有些采用传统研发模式、规模较大、流程较复杂的团队，流动效率甚至不到10%。在能进行准确统计的情况下（指标没有虚高），对很多企业而言，如果流动效率达到30-40%，就已经算是不错的水平了。

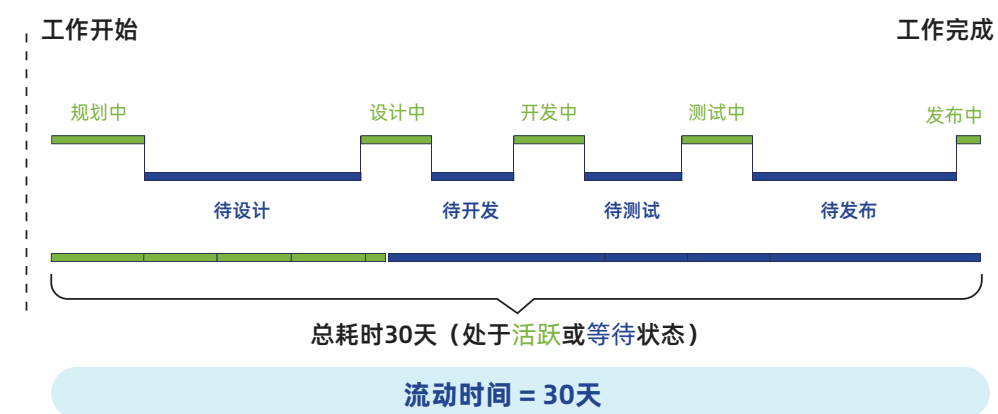
流动时间

贡献人：张乐 晏瑞宇 徐东伟



扫码查看详情

定义



流动时间指从流动项被接受并进入价值流，一直到其完成所花费的时间，包括了工作处于活跃状态的时间和等待状态的时间。

实践出处

《Project to Product》一书

为什么

跟踪流动时间，通过概率思维让交付时间变得更可预测，并相对准确地回答一个核心问题：“工作到底什么时候可以完成？”

何时使用

建议作为日常观察指标。

如何使用

根据研究，交付周期类指标一般符合韦伯分布（Weibull Distribution），所以建议使用85%分位数（而非平均值）来对流动时间进行度量和预测。

指标解读

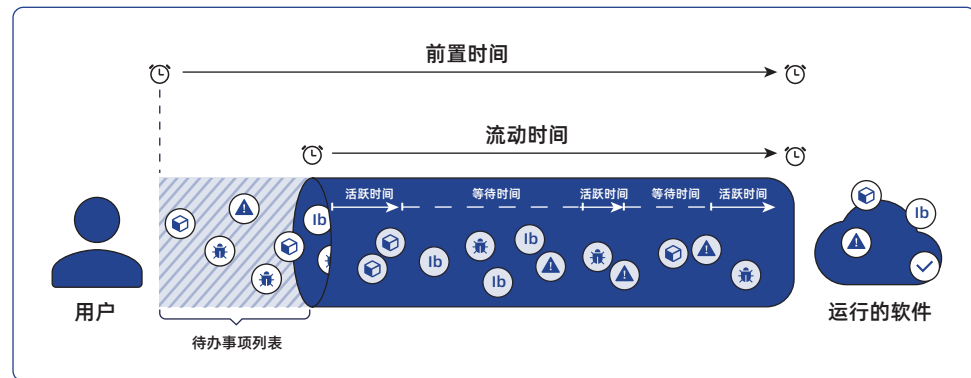
- 数值很低：我们当然希望流动时间不断缩短。但是看到这个数值很低的时候，也别高兴的太早了，要多看看工作有没有被准确跟踪。比如我们在实际工作中，经常出现“后补”需求的情况，比如开发完成到了要上线的时候，因为上线单要关联一个需求单，这个时候再到看板中补上一个需求，然后从第一个阶段直接拖动到最后一个阶

段。类似的情况会导致指标的失真，而指标的准确性是度量的根基，我们需要额外关注。在实践过程中，我们经常观测主指标（如流动时间）的同时，增加一个关于数据健康度的辅助参考指标（如异常数据的比例），以确保主指标的置信度。

- 数值很高，且流动速率很低：有可能是因为在制品过多导致的工作切换，或者工作被阻塞，产生了大量的等待时间，让流动时间被拖长。我们可以结合流动负载和流动效率一起进行更细化的分析。

常见问题解答

1、前置时间、周期时间、流动时间有什么区别？

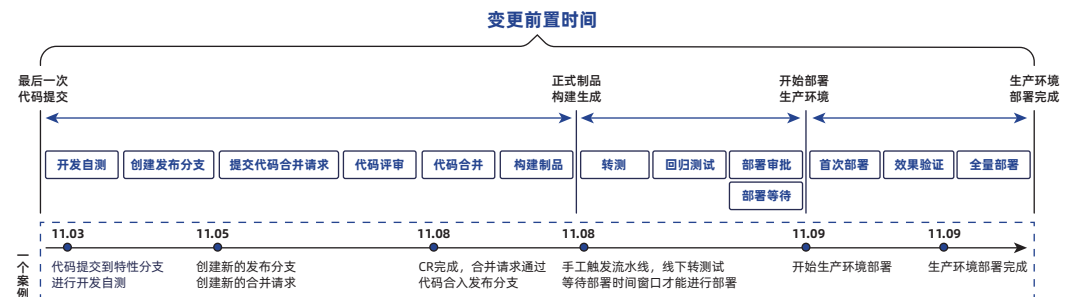


在精益生产中，有两个关键指标用于流程改进，分别是“前置时间”和“周期时间”。前置时间侧重于度量整个流程的时间（工作从“新建”状态开始到“完成”状态之间的时间差），而周期时间则侧重于完成过程中某个步骤所花费的时间（如“开发”阶段的周期时间）。前置时间可以告诉我们端到端流程运行所花费的时间，周期时间可以帮助识别瓶颈（周期时间最长的步骤通常就是瓶颈所在）。

但为了避免混淆，流框架使用了名为“流动时间”的新指标。流动时间始于工作被显式接受（例如新需求评审通过并进入排期）或隐式接受（例如自动升级的事件）的时刻，这与前置时间从工作被提出就开始计时完全不同。流动时间可以作为对产品研发团队交付效率的观测指标，即从确定要做某项工作到完成所需的时间；而前置时间更多是从需求方视角进行观测，即从需求提出到完成所需的时间。

2、为什么没有采用DevOps社区中常用的变更前置时间？

每年的DevOps全球调查报告和DevOps社区中，经常使用名为“变更前置时间”的度量指标，英文是Lead time for changes，即代码提交到部署的前置时间。虽然我们也经常采用这个指标进行效能度量和分析，但它并没有被流框架所采纳。

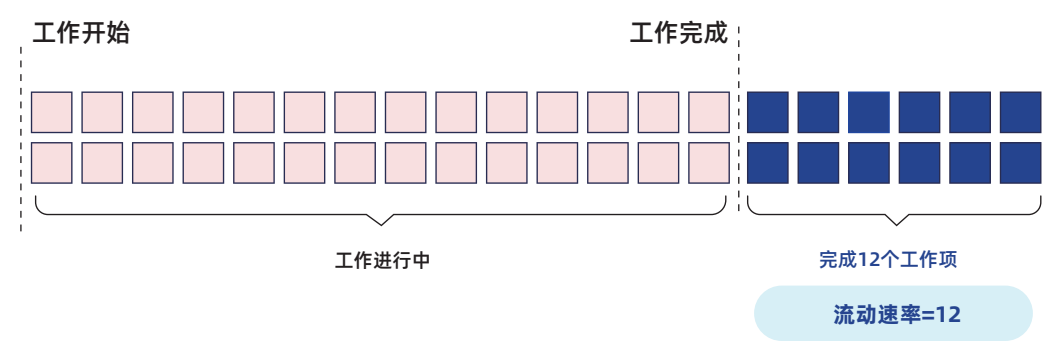


变更前置时间更多的是以开发人员为中心的视角，而不是以客户为中心或以价值流为中心的视角进行设计的，所以它并不足以封装业务价值，虽然是团队工程能力的重要指示器之一，但本质上是更偏局部、更偏过程性的指标。而流动时间的视野更广，观测的是工作项在软件交付管道中端到端的流动，是更偏全局、更偏结果性的指标。

3、流动时间是按自然日来算还是工作日来算？

上文已经讲到，流动时间是一个以客户为中心设计的指标，因此它是以“自然”时间而非“工作日”来进行计算和度量的。

定义



流动速率是指在给定时间内完成的流动项（如需求、缺陷或其他各种类型的工作）的数量，流动速率可用于衡量生产力。

实践出处

《Project to Product》一书

为什么

跟踪流动速率，以有效评估并预测团队可以交付多少工作。

何时使用

建议作为日常观察指标。

如何使用

当流动速率过低时，需要及时调查原因，可能会存在资源紧缺、架构或基础设施等问题，也有可能存在大量等待导致的流动停滞。

指标解读

- 数值升高：一般表明价值交付正在加速；
- 数值降低，且流动时间很长：一般表明交付存在阻塞、依赖，或在制品过多导致的工作切换浪费。

常见问题解答

1、流动速率与敏捷的速率概念有什么区别？

流动速率从敏捷的速率概念改编而来，后者表明了团队在一个时间段内（例如两周的迭代）交付了多少个工作单元（例如故事点数）。但流动速率计算的是在给定时间内完成的流动项的数量，假如一个版本完成了10个需求和5个缺陷，则该版本的流动速率为15。所以，这里的关键区别是，流动速率更简单，它不依赖于对工作量大小、范围或每个流动项优先级的估算。

2、流动速率为什么按需求个数来算，而不是按故事点来算？

诚然，流动项的大小可能会大相径庭，这会让人们倾向于用“故事点”或“T恤”进行估算。但是，使用故事点的估算会容易引发规模冲动（人为的多估算一些），反而可能会更不准确，也经常因此出现业务/产品与研发团队之间对数字的博弈。

所以，流动速率更倾向于用流动项的数量（而非故事点）来进行估算，根据大数定律（如果有足够的试验或实例，事件发生的可能性就是均等的），如果有足够多的流动项，那么在一个时间段内所有流动项都很大，而另一个时间段内的所有流动项都很小的情况，应该很少出现。

另外，对于工作项的合理拆分（如把需求分解为业务需求-产品需求-技术任务），一定程度上也会降低需求颗粒度的差异对指标准确性带来的影响。如果实在不放心，我们在度量流动速率指标的同时，也可以将需求规模（如开发+测试的工作量）作为辅助参考指标一起进行观测。

还需要注意的是，流动速率的度量更适合于跟踪一个价值流内的生产力和交付趋势，而非跨价值流进行比较。

迭代速率

贡献人：顾伟峰 郑乔尹 徐东伟



扫码查看详情

定义

- 迭代速率是对开发小组的速度的度量，现在也有公司称为开发小组产能。可以通过计算开发小组在某次迭代中完成的用户故事点总和来得到速率。
- 如果小组在上一次迭代完成了10个故事点，那么在小组的人数、工作时间、工作效率都不变的情况下，下一次迭代也将完成10个故事点。
- 如果把需求池中的所有故事点进行汇总，就可以通过故事点的总和除以速率来预测迭代次数，以此来规划项目时间。

实践出处

2000年：“速率”一词是相对较晚加入到极限编程中的，它取代了之前被认为过于复杂的“负载系数（load factor）”的概念。

2002年：Scrum社区开始采用测量“速率”的做法。

为什么

使用速率有三个重要的目的：

- 首先，将需求池中故事点的总数除以平均团队速率，可以计算出需要多少个冲刺才能完成这些需求，有利于输出项目时间表。
- 其次，团队可以使用速率作为诊断性指标，用来评估和提高交付能力。通过观察速率值变化，可以评估至少两方面：1) 需求是否被高估；2) 开发效率是否稳定。
- 再次，可以帮助进行迭代规划，预测本迭代团队能够完成的工作量。

何时使用

- 迭代前期：需求规划、项目规划；
- 迭代中期：进度监控；
- 迭代后期：观察数据，调整下次迭代的需求规模和开发资源。

如何使用

计算速率范围

为便于做计划，速率往往是一个范围而非一个固定值。比如，该团队2周一次迭代，速率上下限为25~30个故事点。使用范围可以使我们准确但同时又不至于过分精确。

得到了团队的速率后，我们可以回答如下问题：

- 我们什么时候做完？
- 做几次迭代？
- 我们能完成多少需求？

预测速率

如果团队较为稳定，那么可以使用平均速率用来规划项目。如果团队不稳定，或者团队是新建的，那么就需要预测。

第一个迭代可以根据经验进行主观猜测，完成第一个迭代之后，就可以根据第一个迭代的实际速率来计划第二个迭代。

在3-5个迭代之后，团队的速率就会相对稳定。

输出物

团队迭代速率

参考资料

- 1、Cohn M. 敏捷估计与规划[M]. 北京: 清华大学出版社,2007.
- 2、Rubin K. Scrum精髓: 敏捷转型指南[M]. 北京: 清华大学出版社,2014.
- 3、McDaid K, Greer D, Keenan F, et al. Managing uncertainty in agile release planning[C]//SE-KE. 2006: 138-143.
- 4、Saleh S M, Huq S M, Rahman M A. Comparative study within Scrum, Kanban, XP focused on their practices[C]//2019 International Conference on Electrical, Computer and Communication Engineering (ECCE). IEEE, 2019: 1-6.
- 5、<https://www.agilealliance.org/glossary/velocity/>

平均故障修复时长

贡献人：晏瑞宇



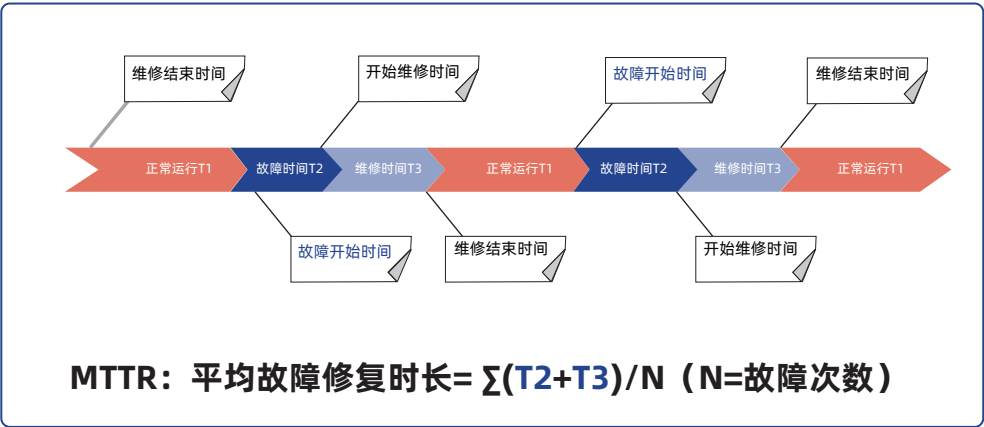
扫码查看详情

定义

平均故障修复时长（MTTR）是修复系统故障并将其恢复到完整功能所需的平均时长。指标反映了线上故障恢复能力，通过缺陷快速修复或回滚降低平均故障修复时长是改进的重点之一。

如何使用

计算方法：平均故障恢复时长（MTTR）= $\Sigma(T2+T3) / N$ （故障次数）



计量单位：时长；

涉及阶段：运营；

补充说明：

平均故障修复时长（Mean Time To Repair）指标越低，故障恢复的越快，故障持续的时间越短，意味着服务更稳定。

这个指标提示我们：在已经发生故障的情况下，如何才能让服务更快恢复？

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

平均故障间隔时长

贡献人：晏瑞宇



扫码查看详情

定义

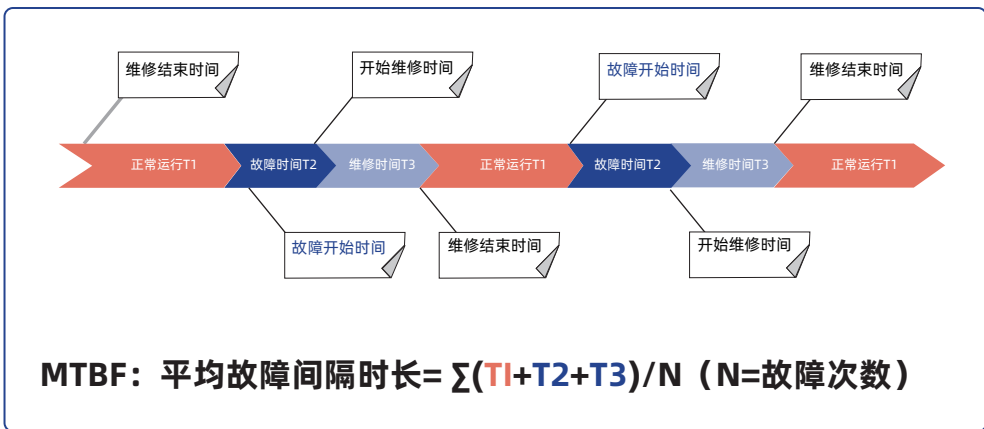
平均故障间隔时长（MTBF）是在正常运行过程中，系统两次相邻故障之间的平均时长。

为什么

平均故障间隔时长的指标反映了线上系统的可靠性，有助于预测系统在下次计划外故障发生之前可以运行多长时间。

如何使用

计算方法：平均故障间隔时长（MTBF）= $\Sigma(T1+T2+T3) / N$ （故障次数）；



计量单位：时长；

涉及阶段：运营。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

线上故障数

贡献人：晏瑞宇



扫码查看详情

定义

线上故障数是系统运行时出现的故障数量。指标反映了系统质量水平。线上故障宜按故障类别、级别进一步分析和改进。

如何使用

计算方法：线上故障数量；

计量单位：#；

涉及阶段：运营；

补充说明：基于行业特征，线上故障一般分为软件故障与硬件故障等。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

系统可用性

贡献人：晏瑞宇



扫码查看详情

定义

系统可用性是在统计范围内系统可用时长占系统全部运行时间的比例。

指标反映了系统的质量和可靠性，有利于衡量设计、开发和运维的质量。

如何使用

计算方法： $\text{系统可用性} = \text{系统可用时长} / \text{系统全部运行时长}$ ；

计量单位：%；

涉及阶段：运营。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

发布成功率

贡献人：晏瑞宇



扫码查看详情

定义

发布成功率是在统计范围内成功发布次数占发布总次数的比例。指标反映了发布过程和脚本质量及生产环境的一致性、稳定性。

如何使用

计算方法：发布成功率 = 成功发布次数 / 发布总次数；

计量单位：%；

涉及阶段：发布；

补充说明：基于企业业务实际情况，可进一步计算一次发布成功率，有利于提升发布流程和发布脚本的质量和效率。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

测试缺陷密度

贡献人：晏瑞宇



扫码查看详情

定义

测试缺陷密度是测试时发现的单位代码规模的缺陷数。指标反映了测试的结果、用于评估开发的质量。

通常，测试缺陷可按缺陷类别、级别进一步分析。

如何使用

计算方法：测试缺陷密度 = 测试检出缺陷数量 / 代码规模；

计量单位：个/代码规模；

涉及阶段：测试；

补充说明：基于企业业务实际情况，可将测试缺陷密度进一步扩展到组件测试、集成测试、验收测试等。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

用例评审通过率

贡献人：晏瑞宇



扫码查看详情

定义

用例评审通过率是在统计范围内，评审通过的测试用例数占提交评审的测试用例总数的比例。指标反映了编写测试用例和理解需求的质量，有利于减少返工和提高交付速率。

如何使用

计算方法：用例评审通过率 = 通过评审的测试用例数 / 提交评审的测试用例总数；

计量单位： %；

涉及阶段：需求、开发、测试；

补充说明：基于企业业务实际情况，可进一步计算测试用例评审的一次通过率，有利于提升用例编写和理解需求的质量。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

用例评审缺陷密度

贡献人：晏瑞宇



扫码查看详情

定义

用例评审缺陷密度是在统计范围内测试用例评审发现的缺陷与测试用例数的比例。指标反映了测试用例的质量，可按缺陷类型、严重程度进行统计，宜注意和软件缺陷进行区分。

如何使用

计算方法：测试用例评审缺陷密度 = 测试用例评审检出缺陷数 / 测试用例数；

计量单位：个/用例数；

涉及阶段：需求、开发、测试。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

提测成功率

贡献人：晏瑞宇



扫码查看详情

定义

提测成功率是在统计范围内，由开发提交的满足测试准入条件的版本的比例。指标反映了开发质量和开发过程的规范性，提高提测成功率利于减少返工、减少测试周期和提高交付速率。

如何使用

计算方法：提测成功率 = 提测成功次数 / 提测总次数；

计量单位：%；

涉及阶段：开发、测试。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

代码评审轮数

贡献人：晏瑞宇



扫码查看详情

定义

代码评审轮数是一次代码合并或提测经过的代码评审的轮数。指标反映了代码合并或提测的质量。控制代码评审轮数有利于提高代码开发者的责任心和和质量意识。

如何使用

计算方法：代码评审轮数；

计量单位：#；

涉及阶段：开发；

补充说明：一轮评审指评审人根据代码做出接受的判断或者提出需要修改的问题。每次修改、补充提交并再次进行评审，轮数加一，直至代码被合并或拒绝。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

代码走查缺陷密度

贡献人：郑乔尹



扫码查看详情

定义

代码走查缺陷密度是代码走查发现的单位代码规模的缺陷数。指标反映了代码质量及走查效果，有利于及早发现缺陷，减少返工。

如何使用

计算方法：代码走查缺陷密度 = 代码走查检出的缺陷数 / 代码规模；

计量单位：个/代码规模；

涉及阶段：开发、测试；

补充说明：代码走查主要由团队执行。发现的缺陷可按缺陷类型和严重程度等统计。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

圈复杂度

贡献人：晏瑞宇 徐东伟



扫码查看详情

定义

圈复杂度是软件代码中独立线性路径的数量。指标反映了系统中控制或信息流的复杂度，有利于降低测试和维护的难度。

如何使用

计算方法：

- 方法一：点边计算法，计算公式为： $V(G) = E - N + 2$
其中，E 表示控制流图中边（Edge）的数量，N 表示控制流图中节点（Node）的数量。
- 方法二：节点判定法，计算公式为： $V(G) = \text{判定节点数} + 1$
判定节点指包括条件的节点。
判定节点的数量再加上 1 即控制流图的区域数。
- 方法三：区域数法，计算公式为： $V(G) = R$
其中 R 代表平面被控制流图划分成的区域数。

计量单位：#；

涉及阶段：开发；

补充说明：可以用来评估一个系统中控制或信息流的复杂度，很多工具可自动测量。高复杂度通常是高缺陷率的领先指标。具有高复杂性的组件通常需要额外的评审，增加测试或重构。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

代码重复率

贡献人：郑乔尹



扫码查看详情

定义

代码重复率是在统计范围内，重复的代码量占代码总量的比例。指标反映了代码的可维护性。

重复代码宜进行抽象和复用，以利于减少缺陷，更快地进行交付。

表达相同或相似语义的代码称为重复代码。

如何使用

计算方法：代码重复率 = 重复代码的代码量/代码的代码总量；

计量单位：%；

涉及阶段：开发；

补充说明：

- 重复代码短期便利了交付，但长期可维护性差，导致修改的工作量成倍增加，甚至导致缺陷。
- 相同或相似逻辑的代码宜进行抽象和复用，方式包括函数、类库或服务。
- 代码重复的判断依赖克隆检测技术，可只在源代码级别按字面判断，也可通过深度代码分析进行语法语义层面的判断。代码量的统计口径可采用代码行数，也可采用代码当量等。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

设计评审通过率

贡献人：晏瑞宇 徐东伟



扫码查看详情

定义

设计评审通过率是在统计范围内，评审通过的组件数占提交评审的组件总数的比例。指标反映了组件及接口定义和设计的质量，利于减少返工和提高交付速率。

如何使用

计算方法：设计评审通过率 = 通过评审的组件数 / 提交评审的组件总数；

计量单位：%；

涉及阶段：设计；

补充说明：基于企业业务实际情况，可进一步计算设计评审的一次通过率，有利于提升组件及接口定义和设计的质量。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

设计评审缺陷密度

贡献人：晏瑞宇



扫码查看详情

定义

设计评审缺陷密度是在设计评审中发现的单位设计规模的缺陷数。指标反映了设计评审的效果。

如何使用

计算方法：设计评审缺陷密度 = 设计评审检出缺陷数 / 设计规模；

计量单位：个/设计规模；

涉及阶段：设计；

补充说明：设计规模可按需求数、功能点、故事点、组件数等进行统计。设计评审缺陷可按缺陷类别、级别进一步分析，以利于针对性地改进。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

需求评审通过率

贡献人：晏瑞宇



扫码查看详情

定义

需求评审通过率是在统计范围内，评审通过的需求数占提交评审的需求总数的比例。

为什么

需求评审通过率的指标反映了需求定义和需求理解的质量，利于减少返工和提高交付速率。

如何使用

计算方法：需求评审通过率 = 通过评审的需求数 / 提交评审的需求总数；

计量单位：%；

涉及阶段：需求；

补充说明：基于企业业务实际，可进一步计算需求评审的一次通过率，有利于提升需求定义和需求理解的质量。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

需求评审缺陷密度

贡献人：晏瑞宇



扫码查看详情

定义

需求评审缺陷密度是在需求评审中发现的单位需求规模的缺陷数。指标反映了需求评审的效果。

如何使用

计算方法：需求评审缺陷密度 = 需求评审检出缺陷数 / 需求规模；

计量单位：个/需求规模；

涉及阶段：需求；

补充说明：需求规模可按需求数、功能点、故事点等进行统计。需求评审缺陷可按缺陷类别、级别进一步分析，以利于针对性地改进。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

演示频率

贡献人：晏瑞宇



扫码查看详情

定义

演示频率是单位时间或版本演示的次数。指标反映了与需求方进行持续沟通并获取反馈的程度，有助于拥抱变化、降低风险。

为什么

演示包括产品演示、原型演示、样品/样机/样件演示，系统演示/成品演示等，是有效沟通、获取反馈、提升客户体验的有效手段。

如何使用

计算方法：演示频率 = 演示次数/时长或版本数；

计量单位：次/时长或版本；

涉及阶段：需求、设计、开发、测试、发布。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

测试一次通过率

贡献人：晏瑞宇



扫码查看详情

定义

测试一次通过率是在统计范围内，由开发提交测试的版本一次性通过测试的比例。该指标反映开发质量，测试一次通过率的提高有利于减少返工和提高交付速率。

如果开发能够通过单元测试或自测等机制，尽量保证提测的质量，有利于减少开发和测试整体的工作量，并提高交付速率。

为什么

企业通过关注测试一次通过率，可避免多次反复提测，浪费资源、延迟交付。

如何使用

计算方法：测试一次通过率 = 一次性通过测试的版本数 / 测试的总版本数；

计量单位：%；

涉及阶段：开发、测试；

补充说明：统计维度可包括产品、项目、团队、周期等。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

平均服务恢复时间

贡献人：王连杰 徐东伟



扫码查看详情

定义

平均服务恢复时间（Mean Time to Recovery, 或 Mean Time to Restore, MTTR）指的是从检测到事件发生到目标系统或组件再次可供用户使用的平均经过时间，它是衡量组织的系统、设备、应用程序和基础设施的可维护性以及发生在IT事件时，恢复IT服务效率的关键指标。

注意：

平均服务恢复时间有别于另一个常用指标——平均故障修复时长（Mean Time To Repair, 也简称MTTR

- 平均故障修复时长：表示修复对象所需的时间；
- 平均服务修复时间：表示修复对象后恢复服务所需的时间。

实践出处

通常认为，平均服务恢复时间（MTTR）来源于服务协议（Service-Level Agreement,SLA）中的关键指标之一。

最早始于1980年代后期，由宽带服务运营商在其提供的服务等级水平中使用。

如今，随着SLAs在其他行业中的广泛使用，MTTR的应用范围也扩展到金融、工商业、服务业以及IT界的物联网、云计算等各个不同领域。

何时使用

系统或组件服务发生故障时。

如何使用

MTTR计算方法

MTTR的计算方法是，以月为单位，将故障导致的总停机时间除以故障总数。

例如，如果一个系统每月发生3次故障并且这些故障导致总共6小时的停机时间，则MTTR为2小时。

$$\text{MTTR} = 6\text{小时} / 3\text{次故障} = 2\text{小时}$$

根据故障的严重程度，修复可能需要几分钟甚至几天的时间，但IT系统的MTTR通常以小时表示。

计算MTTR时需要记住的两个假设：

- 1、通常，每个故障实例的严重程度都不同，因此虽然有些故障需要几天才能修复，但其他故障可能只需要几分钟。因此，MTTR为您提供了预期的平均值。
- 2、重要的是，每个故障实例都应合格且经过适当培训的维护人员按照标准化流程进行处理。这确保了结果的可靠性。

MTTR计算考虑了从事件开始到系统或组件恢复服务之间的时间段。一般包括如下内容：

- 1、通知运维人员；
- 2、诊断问题；
- 3、解决问题；
- 4、重新组装、校准和验证服务；
- 5、重置、测试和启动系统或组件。

MTTR的四个阶段

- 1、识别-从发生故障到技术人员意识到问题的时间段。自动探测和警报系统之类的东西是缩短MTTR识别时间的好方法。
- 2、探明-确定故障后但开始维修之前的时间段。找出或诊断问题通常是MTTR中最耗时的部分。
- 3、修复-实际修复问题所需的时间。通过标准化流程来指导负责解决问题的训练有素的技术人员，可以减少解决问题所需的时间。
- 4、验证-确保应用的修复确实有效所需的时间。实时监控系統是一种有用的工具，可以快速收集数据和报告以显示修复工作正常。

诊断故障原因是MTTR最耗时的的工作。事实上，据统计，80%的MTTR用于找出导致系统或组件失败的原因。记录、管理和拥有系统设备分类账目，包括维护计划、维修/更换组件以及设备监控系统的历史记录，这对于快速缩小可能的故障原因至关重要。在发生故障的情况下，在拨打电话、召开会议和发生错误诊断时会浪费关键时间，从而导致修复失败。

在相同的故障场景中，拥有适当的文档和资产历史记录可以让您快速检查可能导致故障的所有因果因素。管理人员可以查看维护日历以了解设备是否一直得到维护，查看设备上次维修或更换部件的时间，并查看该特定设备过去在哪里出现问题。

输出物

- 直接输出物：修复后的系统或组件服务；
- 间接输出物：业务服务水平以及客户服务质量。

常见问题解答

如何缩短MTTR？

如果MTTR很长，导致原因会因组织而异（需要对实际IT流程和程序进行详细评估），但是通过以下6个步骤，去缩短MTTR似乎对任何企业都有用。

1、了解真实事件

为了缩短MTTR，您应该做的第一件事是对真实事件和故障有一个很好的了解。借助现代企业软件，您可以自动聚合孤立的数据以计算可靠的MTTR，并获得针对这一重要指标的原因分析和宝贵见解。

2、彻底监控

要解决问题，您必须及时识别它，越早发现，越有效。使用合适的监控解决方案，有关系统性能的实时数据始终可以无缝获取，并且在易于理解的仪表盘界面中一揽子显示。通常情况下，如果问题持续存在，您将收到警报通知。

3、制定行动计划

资源不足的小型组织通常需要灵活，但大公司应遵循更严谨的程序和惯例。因此，通常需要使用传统的ITSM技术明确定义角色和响应。经历过全面数字化转型的公司可能具有更大的灵活性，例如使用跨部门协作工具为每个事件构建个性化响应。无论您的预案是什么，都应明确在事件发生时通知谁、如何记录事件以及您的IT团队在着手解决问题时将采取哪些步骤措施等等

4、自动化事件管理机制

为了缩短MTTR，快速响应的第一步是确保正确的人员快速获得有关问题的准确信息。如果在营业时间内发生低优先级事件，您可以致电负责人。但是如果网站在周五晚上8点钟宕机了怎么办？定位相关负责人并手动逐个联系他们太耗时了。这时候自动化事件管理系统可以通过多种渠道（包括电话、微短信和电子邮件等等）一次向所有指定收件人发送警报，从而为您节省大量时间。

5、指定响应团队和角色

明确定义角色和职责对于有效的事件管理和缩短MTTR至关重要。组织类型取决于不同企业的业务需求，但ITIL提供了一个具有以下角色的框架。

- 事故经理：推动事件管理流程并根据需要进行调整和改进的角色。ITIL表示，中小型组织基本上可以将此角色分配给服务经理兼任，而大公司则可以将其设置为单独的角色。事故经理主要负责管理事故管理系统和实施事故管理流程。此外，他还是响应团队的负责人，向管理层报告KPI（关键绩效指标），并管理初级和次级支持。
- 主要（1级）支持：此角色负责给最终用户提供点到点联络从而报告服务中断。负责对事件进行分类并努力尽快恢复失败的服务。主要支持角色无法解决的事件，必须从主要支持小组转发给适当的二级支持人员，并跟踪监控补救活动，实时向用户通知事件的进展状态。
- 二级（2级）支持：二级支持技术人员通常比一级支持更有临场经验。因此，他们将要处理主要支持们无法解决的事件。二级支持响应人员还负责与第三方软件和硬件供应商进行交互，以快速恢复正常的服务。
- 在更为庞大、复杂跟敏感的系统环境中，可能还需要一个具有更高级技能和知识的三级（3级）支持小组。

没有必要严格按照以上框架建立事件响应团队。但是，无论如何组织这个团队，指名一名负责监督事件响应并与团队内外干系人密切沟通的领导，并且所有团队成员都清楚自己的职责，这是必需的。

6、对多个角色的团队成员进行交叉培训

专业专家的存在对事件团队至关重要。然而，大材小用般的依靠这些专家来解决相对琐碎的问题可能会让人不知所措，专家将因其而导致主要职责表现不佳，最终会精疲力尽。更为严重的是，事件发生时如果没有专家，可能会伤害响应团队。

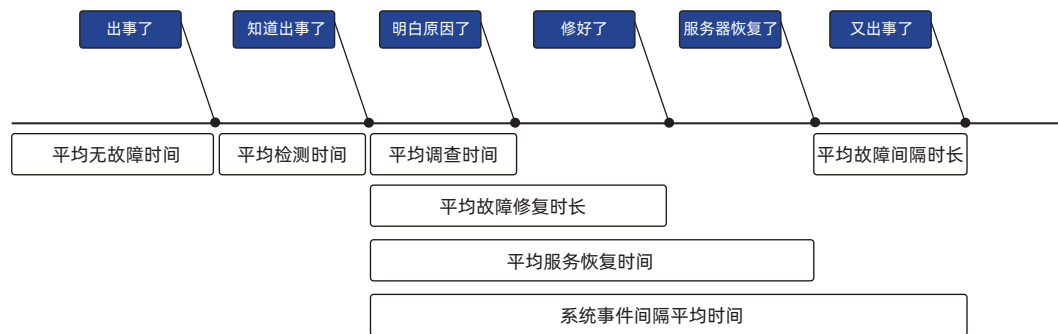
还有哪些其他常用的故障指标？

- 平均检测时间(Mean Time To Detect, MTTD)：从发生问题到检测到问题的平均时间。MTTD指示的时间段是直到IT团队收到故障单，MTTR的测量从这一时间点开始。
- 平均调查时间(Mean Time To Investigate, MTTI)：从发现IT事件到组织开始调查原因和解决方案之间的平均时间。
- 平均故障修复时长(Mean Time To Repair, MTTR)：修复和恢复故障系统所需的平均时间。MTTR是衡量可修复组件和服务的可维护性的指标。根据目标设备和问题的复杂性，MTTR可以是分钟、小时或天的值。
- 平均服务恢复时间(Mean Time to Restore Service, MTRS; Mean Time To Recovery 或者 Mean Time To Restore, MTTR)：从检测到事件到目标系统或组件再次可供用户使用的平均经过时间。(本文阐述对象)
- 平均故障间隔时长(Mean Time Between Failures, MTBF)：一个设备或系统从发生故障到发生下一个故障的平均正常运行时间。它用于预测系统和组件的可靠性和可用性。通过跟踪系统和组件故障之间的正常运行时间

来计算该指标，以便及时维修或者更换系统组件。

- 平均无故障时间(Mean Time To Failure, MTTF): MTTF是设备或系统出现故障之前的平均预期正常运行时间。IT团队通常会花费数天或数周的时间观察系统组件以收集这些数据。与MTBF类似，但MTTF通常用于可更换项目，例如磁带阵列备份驱动器；而MTBF常用于可修复以及可更换的项目。
- 系统事件间隔平均时间(Mean Time Between System Incidents, MTBSI): 两次连续事件之间的平均经过时间。一般通过将MTBF和MTRS(MTBSI=MTBF+MTRS)相加来计算。

故障指标示意图:



参考资料

- 1、[https://cio-wiki.org/wiki/Mean_Time_to_Repair_\(MTTR\)](https://cio-wiki.org/wiki/Mean_Time_to_Repair_(MTTR))
- 2、https://en.wikipedia.org/wiki/Mean_time_to_repair
- 3、<https://www.fiixsoftware.com/maintenance-metrics/mean-time-to-repair-maintenance/>
- 4、<https://www.atlassian.com/incident-management/kpis/common-metrics>
- 5、<https://www.reliableplant.com/mttr-31713>

缺陷逃逸率

贡献人: Becca



扫码查看详情

定义

缺陷逃逸率, Defect Escape Percentage, 简称DEP（也有称为Defect Escape Rate, DER）。

缺陷逃逸率是衡量产品质量的关键指标之一。是指软件产品发布后发现的缺陷数量与该软件产品在整个生命周期发现的所有缺陷数量的比率，通常用来衡量软件团队对软件质量控制水平。

缺陷逃逸率的计算公式一般是：

$$\text{缺陷逃逸率 (DEP)} = [R2 / (R1 + R2)] * 100\%$$

其中：

- R1指的是产品发布前发现的缺陷数；
- R2是产品发布之后所发现的缺陷数。

缺陷逃逸率越低，说明产品总体质量较好，团队对质量控制的水平较好。

造成缺陷逃逸的主要原因主要有：

- 需求理解不充分、不正确，或需求变更频繁但信息更新不及时；
- 测试能力不足，测试用例不全面，测试执行不到位；
- 自动化测试能力弱，难以进行持续测试；
- 测试时间不足，测试计划和策略不合理；
- 测试环境和数据与生产环境有差异；
- 缺少异常测试、探索性测试等。

为什么

如何评估产品的总体质量是否在合适的平衡范围内呢？缺陷逃逸率是非常有效的关键指标，用来跟踪生产环境上的缺陷，不断评估产品质量。

如果在发布后的生产环境中发现了太多问题，则说明在质量内建和测试等方面做得不够好，可能需要放缓进度，改进质量管理、偿还技术债、调整测试策略、增强测试能力、改善质量工具等等，待改进后再继续加快步伐。

通过监测缺陷逃逸率，分析缺陷逃逸率的变化趋势，可以获得产品质量和团队工作情况的持续反馈循环，从而进行改善。

何时使用

产品发布后，如果出现了线上缺陷，就可以计算出缺陷逃逸率（研发阶段也需要记录所有缺陷，才能得到总数）。

如何使用

根据缺陷逃逸率的计算公式

缺陷逃逸率 = 产品发布之后所发现的缺陷数 / 产品发布之前和之后的总缺陷数 * 100%

- 1、在产品研发阶段，需要记录所有缺陷；
- 2、在产品发布后，记录生产环境报告的新缺陷；
- 3、使用以上公式即可得出缺陷逃逸率。

周期性的对缺陷逃逸率进行统计和趋势跟踪

软件行业内，缺陷逃逸率一般在7%~10%，小于7%算优秀，大于10%则比较差（仅供参考，请根据自己公司的实际情况制定目标和改善计划）。

通过一段时间跟踪缺陷逃逸率的变化趋势，可以判断质量改善的效果。

还可能需把生产环境的缺陷级别进行分类

比如致命缺陷、严重缺陷、一般缺陷、轻微缺陷、改进建议，它们的权重是不同的。

另外，也可能需要区分功能模块的特点，比如核心和常用功能，用户使用频繁，很可能发现的问题较多；而不常用的功能，客户报告问题很少，并不代表其质量一定好。

所以对缺陷逃逸率的评估需要结合各方面实际情况进行合理分析。

输出物

当前的缺陷逃逸率；

周期分析缺陷逃逸率的趋势变化。

常见问题解答

使用缺陷逃逸率有哪些注意事项？

- 1、不需要一味追求缺陷逃逸率为0

测试是不可能穷尽的。

测试可以发现存在的问题，但不能证明系统中不存在问题。

同时，产品需要在有限的时间和资源内交付，只能优先确保不要出现致命或严重的线上缺陷。根据功能模块的重要性，测试策略也会根据不同情况有所调整。

因此，上线后出现一些测试过程中没有发现的缺陷是正常的，缺陷逃逸率不太可能为0，而且会发生动态变化。

- 2、缺陷逃逸率的考核始终是一种事后检查，对已经造成的损失于事无补

产品的质量管理更需要防患于未然，尽早介入和预防，在各个环节都需要做好质量内建，不应该只依靠最后阶段的测试来把控质量关。质量不是测出来的，是生产的时候就存在的。

- 3、不要只使用单一指标，也不要指标进行绩效考核

与考核挂钩的指标很容易有副作用，尤其是只看单一指标的情况。很可能出现为了让指标好看而产

生不合理行为，影响指标的真实性，甚至给产品开发带来更大负担或反作用，反而违背了这项指标的初衷。

指标是用来识别现状、给出改进方向，评估改善效果的。

综合使用多个指标以消除单一指标的局限性，例如，综合考虑缺陷逃逸率、缺陷数量和严重程度、周期时间/前置时间、部署频率、客户满意度等指标，以衡量产品交付能力和质量“快与好的平衡”。

参考资料

- 1、<https://www.javamall.com.cn/information/content/news-1739.html>
- 2、<https://baijiahao.baidu.com/s?id=1712368079403299433&wfr=spider&for=pc>
- 3、<https://blog.csdn.net/duzilonglove/article/details/104754101>

代码覆盖率

贡献人：宋晓娜 晏瑞宇



定义

代码覆盖率（Code Coverage）是软件测试中的一种度量，描述程序中源代码被测试的比例和程度，所得比例称为代码覆盖率。

实践出处

代码覆盖是由系统化软件测试所派生的方式。第一份出版的相关参考资料是Miller及Maloney 1963年在ACM通讯上发表的论文。

代码覆盖是飞行设备进行安全认证中的考量项目之一。飞行设备相关认证的指南列在美国联邦航空管理局的文件DO-178B及DO-178C。在诸如IEC 61508或ISO 26262等功能安全标准中并未定义代码覆盖率的需求值，仅仅是将其作为软件测试方法的一个选项，而在对产品有着超高可靠度需求的航空飞行产业标准中（DO-178B）要求代码覆盖率为100%，该数值对于工业或汽车产品实属理想目标。

为什么

目标软件是在特定执行环境下配合特定选项或库所创建的，所执行的每一个函数都会对应到原代码中的功能点。通过代码覆盖，可以测试到目标软件中代码中很少访问甚至从不访问的部分，从而保证代码功能中最重要的条件会被测试到。输出的结果经过分析后可以明确看到代码中哪一部分没有得到执行，并根据实际需求来更新测试代码以确保相应代码得到覆盖。通过代码覆盖与其他测试方法相结合，来开发出一套严格但易于管理的回归测试。

何时使用

一般而言代码覆盖工具及对应的库会影响程序性能，也会消耗服务器内存或其他资源，无法在系统正常使用时测试。因此一般只在开发阶段进行，提供给客户的系统不会包括代码覆盖工具及对应的库。也有一些软件无法用覆盖测试来测试，其覆盖率会用分析的方式来得到近似值，而不是用直接测试到的结果。

如何使用

测试工程师可以根据代码覆盖率工具生成的报告来调整测试用例、输入或是配置，以增加重要功能的代码覆盖率。经常会用到指令覆盖和判断覆盖，前者会报告在测试中会执行到多少比例的代码，后者会报告在测试中会执行到多少比例的的判断结果，二者都会输出一个以百分比表示的覆盖率的度量。其意义则需视进行的代码覆盖种类来决定，通常来说，67%的判断覆盖率会比67%的指令覆盖率会来得全面。

接下来列举几种基本覆盖率准则：

- 函数覆盖率（Function coverage）：有调用到程序中的每一个函数（或副程序）吗？
- 指令覆盖率（Statement coverage）：若用控制流图表示程序，有执行到控制流图中的每一个节点吗？
- 判断覆盖率（Decision coverage）：（和分支覆盖率不同）若用控制流图表示程序，有执行到控制程图中的每一个边吗？例如控制结构中所有IF指令都有执行到逻辑表达式成立及不成立的情形吗？
- 条件覆盖率（Condition coverage）：也称为谓词覆盖（predicate coverage），每一个逻辑表达式中的每一个条件（无法再分解的逻辑表达式）是否都有执行到成立及不成立的情形吗？条件覆盖率成立不表示判断覆盖率一定成立。
- 条件/判断覆盖率（Condition/decision coverage）：需同时满足判断覆盖率和条件覆盖率。

以下是一些可能会用到的覆盖率准则：

- LCSAJ覆盖率：是否执行过每一个LCSAJ（线性代码序列和跳转）？
- JJ路径覆盖率（JJ-Path coverage）：是否执行过每一个JJ路径（从跳转到跳转之间的路径，也就是JCSAJ）？
- 路径覆盖率（Path coverage）：是否执行过程中所有可能的路径？
- 进入点/结束点覆盖率（Entry/exit coverage）：是否执行过函数中所有可能的进入点及结束点？
- 循环覆盖率（Loop coverage）：所有循环是否都有执行过零次、一次及一次以上的测试？
- 参数值覆盖率（Parameter Value Coverage）：对于一个方法的所有参数，是否有执行过其中最常见的数值？

在配合软件开发环境进行代码覆盖率时，需考虑以下几点：

- 1、软件需要通过哪一种代码覆盖？其覆盖率要求是多少？
按照严格程度来看：对应顺序为指令覆盖、条件/判断覆盖及LCSAJ覆盖，越后面的越要求严格。
- 2、是否会通过代码覆盖率来确认受测系统符合某些需求？
- 3、生成的目标代码是否可以追溯到对应的原代码？
如果无法追溯到原代码，DO-178B A等级的认证会要求额外在汇编语言层级进行代码覆盖，以确保所生成代码的准确性。

各种编程语言常见代码覆盖率工具：

- C/C++：GCC + GCov + LCov；
- Java：Maven Cobertura插件、Clover、EMMA、JUnit、Eclipse中的EClemma插件；
- Python：PyUnit + Coverage.py；
- PHP：PHPUnit + Coverage-HTML + XDebug；
- Perl：Test::Class、Devel::Cover；
- Shell：ShUnit2 + Shcov。

常见问题解答

测试过程中代码覆盖率常常被拿来作为衡量测试好坏的指标，甚至用代码覆盖率来考核测试任务完成情况。那代码覆盖率应该达到80%、90%还是100%呢？

实际上，对于大多数项目而言，70-80%的覆盖率较为合理，更高则非常不切实际。单元测试级覆盖率需要比系统级的高10-20%。具体的，代码覆盖率指标的设定需要考虑代码失效的成本、测试相关资源、可测性设计和开发迭代状况等，需要结合具体情况分析，不可一概而论。

参考资料

1、https://en.wikipedia.org/wiki/Code_coverage

2、Joan C. Miller, Clifford J. Maloney. Systematic mistake analysis of digital computer programs. Communications of the ACM (ACM通讯) (New York, NY, USA: 计算机协会). February 1963, 6 (2): 58-63.

3、RTCA/DO-178B, Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics, December 1, 1992.

4、RTCA/DO-178C, Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics, January, 2012.

5、Glenford J. Myers. The Art of Software Testing, 2nd edition. Wiley. 2004. ISBN 0-471-46912-2.

工作量分布



扫码查看详情

贡献人：晏瑞宇

定义

工作量分布是工作量按选定维度的统计值和分布比例。该指标反映企业交付过程中不同维度下的工作投入比例，可支持优化不同维度的工作分配。

维度包括生命周期阶段、需求、组件、版本等。

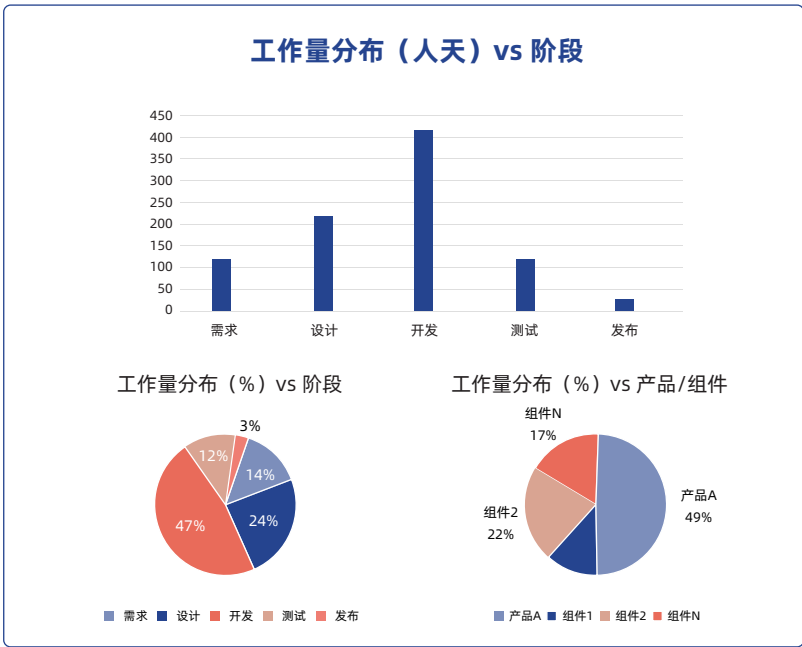
如何使用

计算方法：工作量按不同维度分段进行统计；

计量单位：%；

涉及阶段：需求、设计、开发、测试、发布、运营；

补充说明：工作量分布可按多维度进行统计分析，并可与历史基线进行比较，以识别突出问题，进行管控和改进；亦可与需求吞吐量、代码当量等指标进行关联分析。示例如下：



参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

人员流动率

贡献人：晏瑞宇



定义

人员流动率是统计范围内，离职、调入或调出员工与员工总数的百分比。指标反映了企业组织与员工队伍的稳定性。除了整体计算，宜对骨干员工和新员工进行分类统计。

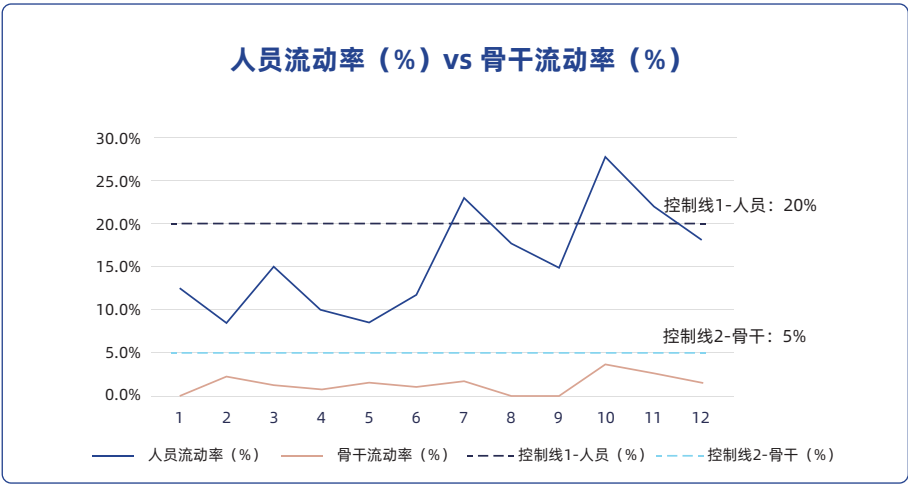
如何使用

计算方法：人员流动率 = 流动员工数/员工总数；

计量单位：%；

涉及阶段：运营；

补充说明：控制人员流动率，特别是将骨干流动率控制在合理水平至关重要，对进度、质量、成本等有显著影响。人员流动率可在项目、产品线、部门各维度进行统计分析，示例如下：



参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

人力成本

贡献人：晏瑞宇



定义

人力成本是软件研发各活动的工作量投入按人员费率折算的总成本。指标反映了软件研发活动的人力投入，取决于不同技能级别的人员费率和工时投入。

有利于企业管理者从人力成本方面对项目支出进行掌握。

如何使用

计算方法：人力成本 = $\sum$ (工时投入*人员费率)；

计量单位：货币；

涉及阶段：需求、设计、开发、测试、发布、运营；

补充说明：人力成本宜按产品和组件（或应用）进行统计，可分配到相应的阶段，以便进一步分析和比较，支持项目策划、控制和改进，示例如下：

产品/组件	需求	设计	开发	测试	发布	合计
产品A	¥ 240,000.00	¥ 360,000.00		¥ 80,000.00	¥ 50,000.00	¥ 730,000.00
组件1			¥ 120,000.00	¥ 20,000.00		¥ 140,000.00
组件2			¥ 180,000.00	¥ 12,000.00		¥ 192,000.00
组件N			¥ 210,000.00	¥ 21,000.00		¥ 231,000.00
合计	¥ 240,000.00	¥ 360,000.00	¥ 510,000.00	¥ 133,000.00	¥ 50,000.00	¥ 1,293,000.00

组织级通常按人员级别明确相应费率，如：需求工程师：3.5 万元/人月，开发工程师 2.5 万元/人月等，以利于计算人力成本。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

技能指数

贡献人：晏瑞宇



扫码查看详情

定义

技能指数是对员工技能评价的分值。指标反映了员工的知识、经验、能力等级，对人员工作分配和成本有显著影响。

宜针对人员所在岗位或领域，如项目管理、需求分析、设计开发、测试等进行评价。企业可设置评价权重和取值范围。

为什么

员工技能指数的统计和分析有助于建立更合理的人员梯队和恰当的分配工作。

如何使用

计算方法：技能指数 = $\sum$ (技能分值*权重)；

计量单位：分值；

涉及阶段：需求、设计、开发、测试、发布、运营。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

非人力成本

贡献人：晏瑞宇



扫码查看详情

定义

非人力成本是直接用于项目的设备、培训、差旅等及分摊费用的总和。指标反映软件研发活动的非人力投入。

如何使用

计算方法：非人力成本 = $\sum$ (各类支出费用)；

计量单位：货币；

补充说明：非人力成本一般按项目、产品线、部门等进行核算和分摊。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

预算执行率

贡献人：晏瑞宇



扫码查看详情

定义

预算执行率是在统计范围内，实际支出资金与实际到位资金的百分比。指标反映了项目资金使用效率和进程。

如何使用

计算方法：预算执行率 = 实际支出资金 / 实际到位资金；

计量单位：%；

涉及阶段：需求、设计、开发、测试、发布、运营；

补充说明：

- 实际支出资金指一定时期（本年度或项目期）内项目实际拨付的资金；
- 实际到位资金指一定时期（本年度或项目期）内落实到具体项目的资金。

宜在项目计划阶段建立预算执行计划，并在执行过程中计算预算执行率。预算执行率越接近100%，显示计划执行越佳；反之，则越差。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

发布时长

贡献人：晏瑞宇



扫码查看详情

定义

发布时长是软件单次发布的时长。指标反映了系统复杂度和基础设施等状况，过长的时间应考虑软件发布过程的优化。

如何使用

计算方法：发布时长 = 发布完成时间 - 发布开始时间；

计量单位：时长；

涉及阶段：发布。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

发布频率

贡献人：晏瑞宇



扫码查看详情

定义

在单位时间内，软件发布的次数。该指标反映了软硬件基础设施、交付过程、研发团队质量和效率等多种能力。

如何使用

计算方法：发布次数/单位时间；

计量单位：次/时长；

涉及阶段：发布；

补充说明：控制合理的发布频率可以满足快速交付和获得客户及时反馈的需要。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

回滚成功率

贡献人：晏瑞宇



扫码查看详情

定义

回滚成功率是在统计范围内回滚成功次数占回滚总次数的百分比。指标反映了回滚脚本质量和基础设施的能力，回滚成功率是降低持续交付风险的关键指标。

如何使用

计算方法：回滚成功率 = 回滚成功次数 / 回滚总次数；

计量单位：%；

涉及阶段：开发、测试、发布；

补充说明：宜考虑针对不同环境的回滚成功率，如开发环境、测试环境、准生产环境、生产环境等，有利于针对性的进行改进。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

部署成功率

贡献人：晏瑞宇



扫码查看详情

定义

部署成功率在统计范围内统一部署成功次数占统一部署总次数的百分比。指标反映了部署脚本质量和部署环境的一致性、稳定性。

如何使用

计算方法：部署成功率 = 统一部署成功次数/统一部署总次数；

计量单位：%；

涉及阶段：开发、测试、发布；

补充说明：宜考虑针对不同环境的部署成功率，如开发环境、测试环境、准生产环境、生产环境等，有利于针对性的进行改进。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

部署时长

贡献人：晏瑞宇



扫码查看详情

定义

部署时长是软件单次部署的时长。指标反映了系统复杂度和基础设施等状况，过长的时间应考虑软件部署的优化。

如何使用

计算方法：部署时长 = 部署完成时间 - 部署开始时间；

计量单位：时长；

涉及阶段：开发、测试、发布。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

部署频率

贡献人：晏瑞宇



扫码查看详情

定义

部署频率是在单位时间内，软件进行部署的次数。指标反映了持续部署的能力，有助于及早发现和修复缺陷以及环境的不一致性。

如何使用

计算方法：部署频率 = 部署次数/单位时间；

计量单位：次/时长；

涉及阶段：开发、测试、发布；

补充说明：宜考虑针对不同环境的部署时长，如开发环境、测试环境、准生产环境、生产环境等，有利于针对性的进行改进。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

环境整备时长

贡献人：晏瑞宇



扫码查看详情

定义

环境整备时长是环境整备所需的时长。指标反映了环境的复杂度和基础设施等状况，过长的时间应考虑环境的优化。

如何使用

计算方法：环境整备时长 = 环境整备完成时间 - 环境整备开始时间；

计量单位：时长；

涉及阶段：开发、测试、发布、运维；

补充说明：在软件研发过程中，开发环境、系统测试环境、验收测试环境的整备时长和一致性，对快速持续交付能力有显著影响。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

测试自动化率

贡献人：晏瑞宇



扫码查看详情

定义

测试自动化率是自动化测试用例占测试用例总数的百分比。指标反映了自动化测试的能力，有助于提高测试效率和快速交付能力。

如何使用

计算方法：测试自动化率 = 自动化测试用例数/总测试用例数；

计量单位：%；

涉及阶段：开发、测试；

补充说明：通常按功能测试、性能测试、安全测试等，以及单元测试、组件测试、集成测试、系统测试等维度进行统一，特别应关注回归测试自动化比例。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

缺陷重开率

贡献人：晏瑞宇



扫码查看详情

定义

缺陷重开率是在统计范围内，重新打开的缺陷占总缺陷的比率。指标反映了缺陷修复的效果，重开率高应触发针对性的改进。

如何使用

计算方法：重新打开缺陷数/总缺陷数；

计量单位：%；

涉及阶段：测试、发布、运维。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

构建成功率

贡献人：晏瑞宇



扫码查看详情

定义

构建成功率是在统计范围内统一编译构建成功次数占统一编译总构建次数的百分比。指标反映了代码质量、编译环境的稳定性。

如何使用

计算方法：构建成功率 = 统一编译构建成功次数/统一编译构建总次数；

计量单位：%；

涉及阶段：开发；

补充说明：构建成功率低可能意味着过程规范性、代码质量、工具链、基础设施等持续交付能力需进行改进。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

构建时长

贡献人：晏瑞宇



扫码查看详情

定义

构建时长是软件单次编译构建的时长。指标反映了系统复杂度和基础设施等状况，过长的时间应考虑编译构建的优化。

如何使用

计算方法：构建时长 = 构建完成时间 - 构建开始时间；

计量单位：时长；

涉及阶段：开发。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

构建频率

贡献人：晏瑞宇 徐东伟



扫码查看详情

定义

构建频率是单位时间内软件进行统一编译构建的次数。指标反映了持续构建的能力，有助于及早发现和修复缺陷。

如何使用

计算方法：构建频率 = 构建次数/单位时间；

计量单位：次/时长；

涉及阶段：开发。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

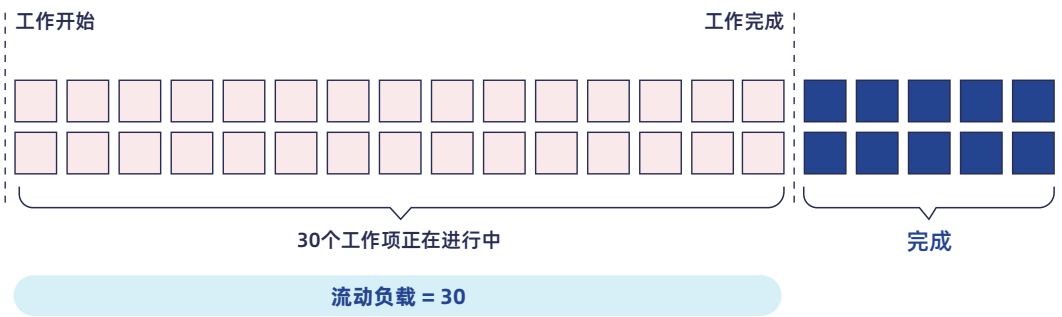
流动负载

贡献人：张乐 晏瑞宇 徐东伟



扫码查看详情

定义



流动负载指价值流中在制品的数量（已开始、未完成，即正在进行中的工作），包含了状态为活跃或等待的流动项的数量。

实践出处

《Project to Product》一书

为什么

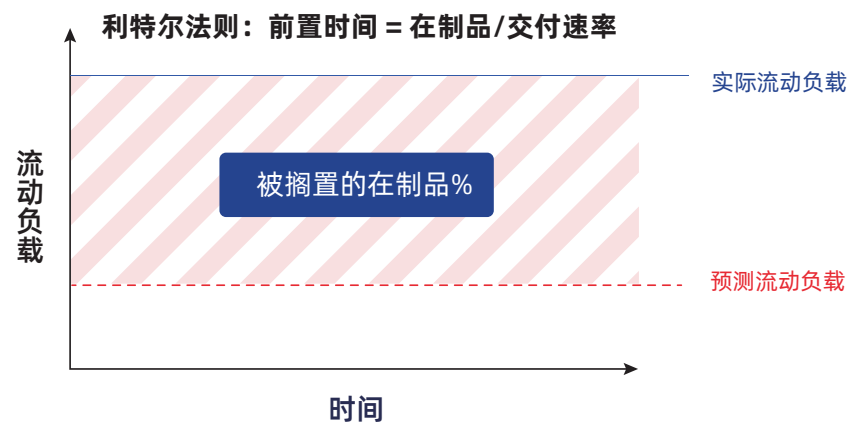
流动负载是一个先导性指标，可以用来发现在制品过多对速度类指标和团队满意度的影响。

何时使用

我们可以通过不断调整和实验，找到产品价值流最优的流动负载，此时流动速率较高，并且流动时间较短。

指标解读：

- **数值较低：**可能只有少量的工作正在完成，可能出现了闲置的情况。
- **数值较高：**过多的流动负载（在制品）很可能导致交付延迟、成本增加、质量下降、员工抱怨，长期超过团队产能安排工作将导致职业倦怠。还可以进一步分解为以下两种情况：
 - **数值较高，并且流动时间很短：**可能有很多的工作被忽略或搁置了，即存在很多“僵尸需求”，一直停留在交付管道中却又因为优先级一直没有时间处理。这时需要清理在制品，评估真正需要完成的工作。如果真的重要就让工作继续及时推进，如果不重要就干脆把工作移出交付管道。
 - **数值较高，并且流动时间很长：**在制品过多导致的工作切换可能是罪魁祸首，过高的流动负载直接影响了交付效率。这时可以采用精益实践，限制在制品并采用拉动模型（如采用精益看板的方法）。可能还要为资源不足的角色/岗位增加容量，或提升自动化的水平。



通过查看系统中指定天数（如10天）没有进展的工作，可以发现系统中的问题和瓶颈，一方面找到并消灭较低价值的“僵尸需求”，减少被搁置的工作；另一方面识别出被阻塞的工作，通过当前阶段及上下游的协同优化，促进它们尽快恢复流动。

特别需要注意的是，根据利特尔法则，流动时间 = 流动负载 / 流动速率，当流动负载的当前值已经高于（流动速率*流动时间）的预测值，则预示未来会有工作无法如期完成。这时就已经发现了未来交付计划及周期的风险，需要对流动时间的预测进行修正，进而实现更准确的承诺/决策，这正是先导性指标的价值所在。

常见问题解答

1、流动负载应该从何处开始计算？尚未开始的工作要计算么？

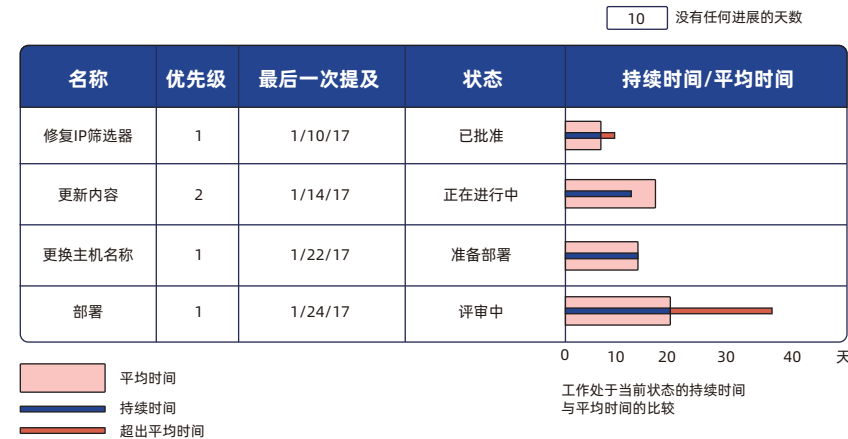
如果将价值流想象为一条管道，其中所有尚未开始或已经完成的流动项都在管道的两端，而流动负载就是管道内正在进行的工作单元数，包括所有部分完成的流动项。当流动负载过大，由于队列时间过长，价值流的过度利用会极大地影响交付速度。

2、流动负载有没有绝对数字来说明好坏？

业界很难有一个绝对的数字来说明流动负载应该是多少，不同业务类型、处于不同发展阶段的团队，对流动负载的承受能力都有很大差别。所以，建议采用实验思维，关注流动负载的高低，导致了流动速率和流动时间怎样的变化，从而找到一个适当的平衡点。追求过高的资源饱和度对产品开发而言并没有任何好处。百分之百资源利用率对于制造业和软件交付都存在同样问题，都会对流动速率和流动时间产生很大的负面影响。

3、流动负载如何进行有效下钻找到具体问题？

可以通过停滞项工作报告，展示在交付管道中有哪些未完成的工作，以及它们在当前阶段已经停滞了多久的时间。



过程符合度

贡献人：晏瑞宇



扫码查看详情

定义

过程符合度是统计范围内体系和成果物审计合规的比例。指标反映了制度和规范的落实情况。

应以质量审计和配置审计为依据，关注项目中体系实施的必要性和充分性，并计算其符合度。

何时使用

适用的过程可根据企业业务实际确定，包括但不限于需求开发、需求管理、版本规划、设计规范、开发规范、测试规范、评审规范、验收规范、上线规范、运营规范、项目立项、项目估算、项目策划、项目监控、项目结项、风险管理、问题管理、缺陷管理、事件管理、度量分析、决策分析、根因分析、配置管理、质量追溯、质量保证等制度和规范及其检查单等。

如何使用

计算方法：过程符合度 = $(\sum(G)*5+\sum(Y)*3)/(\sum(G,Y,R)*5)$;

G/Y/R 是审计评价结果，其中：

- G：大部分满足；
- Y：部分满足；
- R：不满足；
- NA：不适用；
- NY：时机未到。

计量单位：%；

涉及阶段：需求、设计、开发、测试、发布、运营。

补充说明：

示例如下：

适用过程/规范/指南	产品线1				产品线N					
	项目1	项目2	项目3	项目4	项目5	项目6	项目7	项目8	项目9	项目10
工程管理程序	G	G	G	G	G	G	G	G	G	G
需求分析指南	G	Y	G	G	Y	Y	G	G	Y	G
设计规范	G	G	G	G	G	G	Y	G	G	G
集成及测试指南	Y	G	Y	G	G	G	G	G	Y	Y
测试验证规范	G	G	G	Y	G	Y	G	Y	G	G
技术评审规范	G	Y	G	G	G	Y	Y	G	G	G
代码评审规范	Y	Y	R	Y	Y	G	G	G	Y	Y
项目实施流程	G	NA	NA	NY	NA	NA	G	NY	NA	NA
项目管理程序	G	G	G	Y	G	G	G	G	G	G
项目估算指南	Y	Y	Y	G	Y	Y	Y	G	Y	G
风险与机会管理指南	G	G	G	G	G	Y	Y	Y	G	G
项目度量分析指南	Y	Y	G	Y	G	NY	Y	G	Y	Y
决策分析指南	R	Y	Y	NY	Y	NY	G	R	G	NY
根因分析指南	R	R	R	NY	NY	NY	NY	R	NY	NY
项目配置管理程序	G	Y	G	G	G	G	G	G	G	G
项目质量保证程序	G	Y	G	G	G	Y	G	G	G	G
.....										
	78%	72%	79%	88%	89%	80%	87%	81%	86%	91%

术语定义	缩略语	分值
大部分满足	G	5
部分满足	Y	3
不满足	R	0
不适用	NA	
时机未到	NY	

过程符合度数据可进一步汇总到产品线和部门级，并通过时序分析，展示和分析过程符合度的变化，与目标值或基线进行对比分析，支持过程改进及其效果评价。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

审计频率

贡献人：晏瑞宇



扫码查看详情

定义

审计频率是在单位时间内内外部审计的次数。指标反映了外部合规和内部管控的力度。

内外部审计的频率宜按类别适度控制，为体系落地，及早发现问题，及时改进提供支持。

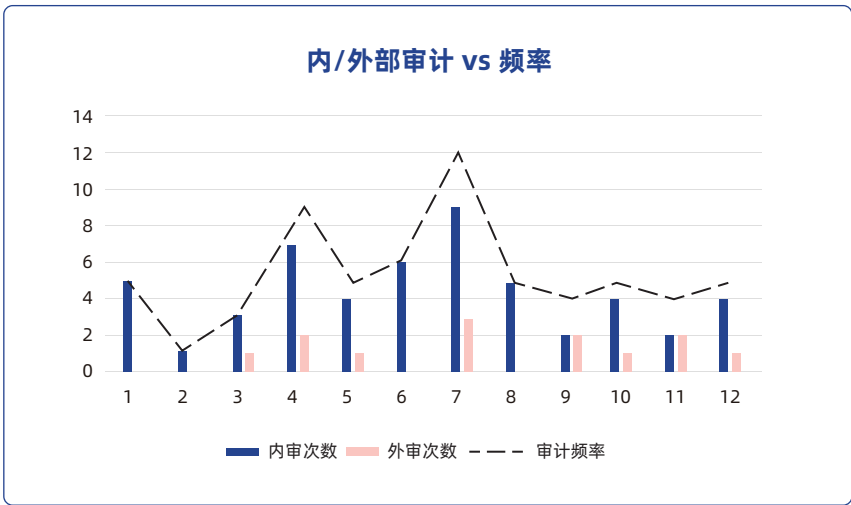
如何使用

计算方法： 审计频率 = (内部审计次数+外部审计次数)/时长；

计量单位： 次/时长；

涉及阶段： 需求、设计、开发、测试、发布、运营；

补充说明： 为了保证质量和合规，特定行业需要频繁进行审计。合理的审计频率，可以保证过程持续置于监管之下，并保证规章制度得以贯彻执行，示例如下：



审计频率的统计宜按类别进行统计，如：外部审计、内部审计、需求审计、设计审计、开发审计、测试审计、发布审计、质量审计等。内部审计宜按季度策划和执行，确保项目、产品线、部分和体系的覆盖率和持续受控。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

审计问题关闭率

贡献人：晏瑞宇



扫码查看详情

定义

审计问题关闭率是统计范围内审计发现问题已关闭的比例。指标反映了及时处理并解决审计所发现问题的能力。

如何使用

计算方法： 审计问题关闭率=已关闭的审计问题数/审计问题总数；

计量单位： %；

涉及阶段： 需求、设计、开发、测试、发布、运营；

补充说明： 宜按类型、严重程度等分类统计，以及时处理和解决内外审计中的关键不符合项。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

专项改进完成率

贡献人：晏瑞宇



扫码查看详情

定义

专项改进完成率是统计范围内专项改进项目的完成比例。指标反映了专项改进在公司级、部门级、产品线的实施情况。

如何使用

计算方法： 专项改进完成率 = 按时完成的专项改进数/计划完成的专项改进数；

计量单位： %；

涉及阶段： 需求、设计、开发、测试、发布、运营；

补充说明： 宜按照年度、季度、月度统计改进项目数、状态、成功率等，以确保相应的持续改进能力。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

改进效果评价

贡献人：晏瑞宇



扫码查看详情

定义

改进效果评价是改进前后选定度量数据的变化比例。指标反映了专项整改的量化的效果。

如何使用

计算方法： 改进效果评价 = 改进后数据/改进前数据；

计量单位： %；

涉及阶段： 需求、设计、开发、测试、发布、运营；

补充说明： 例如，改进前项目返工率均值为35%，改进后返工率均值降到17.5%，则项目返工率均值下降了50%。改进效果评价还可使用其他统计方法，如假设检验（检验均值、标准差、中位数的变化等），以建立对过程改进效果的深入理解，并支持决策分析和风险评估。

参考资料

《软件研发效能度量规范》，中关村智联软件服务业质量创新联盟。

定义

项目健康指数（PHI）是对项目的综合状态进行量化刻画，是多个度量元的汇总结果。

每个组织可以定义自己的度量元和项目健康指数的计算方法。

实践出处

该实践是麦哲思科技在实践中总结的经验。

为什么

使用项目健康指数（PHI），管理者能够从宏观上了解项目的状态，而不是陷入具体的细节中，片面地了解项目的状态，从而也便于进行横向和纵向的对比分析。

何时使用

在整个项目过程中都可以使用。

如何使用

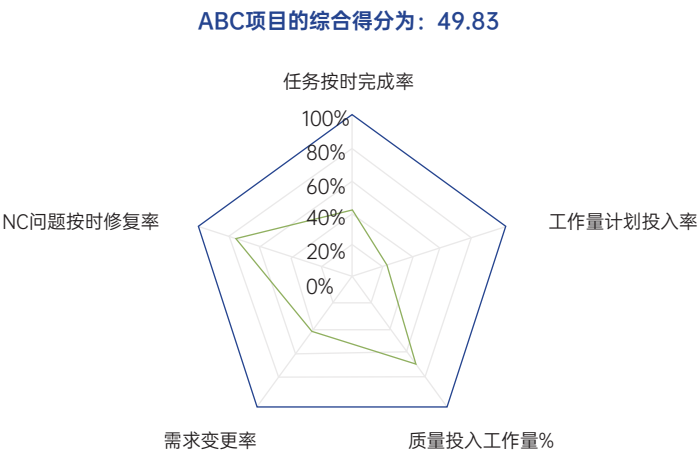
我们假设项目健康指数（PHI）是由如任务按时完成率、质量投入工作量、需求变更率和缺陷按时修复率等因子组成，每个组织可以定义自己的项目健康指数的计算方法。例如，下表为对某项目进行度量后得到数据。

	上限	下限	实际值	规格化上限	规格化下限	实际值得分
任务按时完成率	200%	0%	85%	100%	0%	42.50%
工作量计划投入率	200%	0%	50%	100%	0%	25.00%
质量投入工作量%	30%	0%	20%	100%	0%	66.67%
系求变更率	20%	0%	12%	100%	0%	40.00%
NC问题按时修复率	100%	0%	75%	100%	0%	75.00%
					综合结果	49.83%

注1：NC问题，即不符合项。

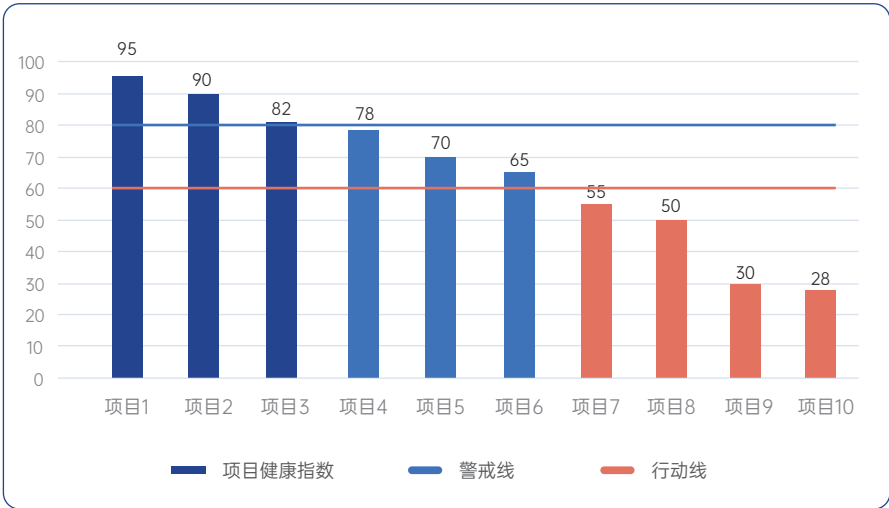
注2：所有的5个指标得分都是越大越好，所有需求变更率在规格化后得分是40%，是1-（12%/20%）计算得到的。

根据上表的数据，选择雷达图建立项目综合指数指示器，如下图所示：



有了每个项目的PHI，就可以在各个项目组之间进行总体状态的对比分析。

例如，某公司确定60分为行动线，当项目的PHI小于60时，高层管理者就需要对该项目采取措施了。如下图：



项目管理

整合管理

进度管理

质量管理

采购管理

利益相关人管理

挣值分析

贡献人：薛健 晏瑞宇 徐东伟



扫码查看详情

定义

Earned Value Analysis (EVA) 挣值分析法，也称为赢得值、获得值或偏差分析法。挣得值分析法是在工程项目实施中使用较多的一种方法，是对项目进度和费用进行综合控制的一种有效方法。

实践出处

1967年美国国防部（DoD）开发了挣值法并成功地将其应用于国防工程中。并逐步获得广泛应用。

为什么

挣值法的核心是将项目在任一时间的计划指标、完成状况和资源耗费综合度量。将进度转化为货币或人工时；如果是工程量，则转化为钢材吨数、水泥立方米、管道米数或文件页数。

挣值法的价值在于将项目的进度和费用综合度量，从而能准确描述项目的进展状态。挣值法的另一个重要优点是可以预测项目可能发生的工期滞后量和费用超支量，从而及时采取纠正措施，为项目管理和控制提供了有效手段。

何时使用

使用挣值分析的时机包括但不限于：

- 到达里程碑节点时；
- 发生重大变更或风险时。

如何使用

基本参数

- PV（Planned Value，计划值），也称为BCWS（Budgeted Cost for Work Scheduled, 计划工作量的预算费用）。
 - PV是指项目实施过程中某阶段计划要求完成的工作量所需的预算费用。计算公式为：PV=计划工作量×预算定额。PV主要是反映进度计划应当完成的工作量(用费用表示)。
 - PV是与时间相联系的，当考虑资金累计曲线时，是在项目预算s曲线上的某一点的值。当考虑某一项作业或某一时间段时，例如某一月份，PV是该作业或该月份包含作业的预算费用。
- AC（Actual Cost，实际成本），也称为ACWP（Actual Cost for Work Performed，已完成工作量的实际费用）。
 - AC是指项目实施过程中某阶段实际完成的工作量所消耗的工时（或费用）。AC主要反映项目执行的实际消耗指标。

- EV（Earned Value，挣值），也称为BCWP（Budgeted Cost for Work Performed，已完工作量的预算成本）。或称盈值和挣得值。
 - EV是指项目实施过程中某阶段按实际完成工作量及按预算定额计算出来的费用，即挣得值(Earned Value)。
 - EV的计算公式为：EV=已完工作量×预算定额。
 - EV的实质内容是将已完成的工作量用预算费用来度量。

四个评价指标

- CV（Cost Variance，费用偏差）
 - CV是指检查期间BCWP与ACWP之间的差异，计算公式为CV=EV-AC。
 - 当CV为负值时表示执行效果不佳，即实际消费费用超过预算值即超支。
 - 反之当CV为正值时表示实际消耗费用低于预算值，表示有节余或效率高。
 - 若CV=0, 表示项目按计划执行。
- SV（Schedule Variance，进度偏差）
 - SV是指检查日期BCWP与BCWS之间的差异，其计算公式为SV=EV-PV。
 - 当SV为正值时表示进度提前。
 - SV为负值表示进度延误。
 - 若SV=0, 表明进度按计划执行。
- CPI（Cost Performed Index，费用执行指标）
 - CPI是指挣得值与实际费用值之比，CPI=EV/AC。
 - 当 CPI > 1表示低于预算。
 - CPI < 1表示超出预算。
 - CPI = 1表示实际费用与预算费用吻合，表明项目费用按计划进行。
- SPI（Schedule Performed Index，进度执行指标）
 - SPI是指项目挣得值与计划值之比，即SPI=EV/PV。
 - 当 SPI > 1表示进度提前。
 - SPI < 1表示进度延误。
 - SPI = 1表示实际进度等于计划进度。

参考资料

- 1、<https://wiki.mbalib.com/wiki/%E6%8C%A3%E5%BE%97%E5%80%BC%E5%88%86%E6%9E%90%E6%B3%95>
- 2、<https://zhuanlan.zhihu.com/p/136275502>
- 3、<https://www.tacticalprojectmanager.com/earned-value-analysis/>
- 4、<https://zhuanlan.zhihu.com/p/376409674>
- 5、《项目计划、进度与控制》

限制在制品

贡献人：栾绍晨 晏瑞宇 徐东伟



定义

在制品（WIP-Work In Progress）是指还处在生产过程中但尚未完成的产品/工作。这些产品/工作可能处在制作过程中，也可能在等候排队。

而限制在制品（Limit WIP）限制了工作流每个状态或全流程中可以存在的最大工作量，通常体现在看板上，而它限制的对象可以是个人、团队、甚至是整个组织。通过限制在制品可以帮助找到团队成员工作流程中效率低下的环节—以消除流程中的浪费。限制在制品使团队能够优化其工作流程以实现价值交付。这也体现了限制在制品的核心逻辑：暂缓开始，聚焦完成。

实践出处

限制在制品的实践源自于丰田的看板系统。后来在大卫·安德森的看板方法中被定义为看板方法的6大实践之一，被广泛推广。

为什么

限制在制品可以带来以下好处：

- 减少工作上下文切换。如果不进行在制品限制，人们激活/开始的任务通常会更多，也会有任务加入会打断正在进行的工作。根据加州大学欧文学院的信息学教授Gloria Mark的调查显示，即使是短暂的中断，人们也平均需要23分15秒才能重新集中注意力。频繁的打断会影响工作的质量，增加潜在风险（例如缺陷的数量），从而形成恶性循环。而需要更多的时间恢复注意力进行产出也会产生更多的浪费。使用限制在制品可以让人更专注于完成已经开始的任务（而不是被其他任务打断或者不停开始新的任务），而高效工作的秘诀就是一次只专注于一件事,进而也会减少因为过多的工作上下文切换导致的浪费。
- 快速识别阻碍，发挥团队的作用。过多的在制品会导致要同时处理更多的任务，也可能会碰到困难的任务就停下来先做别的。会导致单个已开始任务的产生额外的工作。例如一个任务在两个周以前就已经开发一半了，但是因为阻塞没有及时处理，减缓了价值的流动。而过后又继续进行开发时，会发现不得不重新回顾代码逻辑甚至重新编写代码和设计。增加了人力、时间、脑力等等的浪费。而使用限制在制品会更直观地产生流程的迟滞（卡片在白板上移动的非常缓慢）、积压（某列中有很多卡片）甚至完全停止（卡片一直在等待）。因为更少的展示内容会更直观地暴露问题，通过工作流的观察，可以把给更多改进的机会呈现在表面。当有问题暴露出来时，就是开始改善的良好契机。同时，因为可以更直观地发现停滞的卡片，也可以更好地为团队跨职能帮助创造契机。当出现问题时，不是先放下当前问题开始新的工作，而是大家一起解决当下优先级和价值更高的问题，能够更好地发挥团队的作用（因为我们要优先交付高价值内容，所以越早开始的工作价值就应该越高）。
- 缩短交付时间（前置时间）。根据利特尔法则：平均前置时间=在制品数量/平均吞吐率。如果每15分钟可以制作的披萨数量是固定的20个（平均吞吐率），而我们有60个披萨（在制品）在等待制作，则平均的每个披萨的前置时间为45分钟。如果我们限制在制品的数量在40个，则平均的前置时间可以减少到30分钟。顾客等待的时间就会减少。所以限制在制品的数量可以更快地给客户交付价值。

何时使用

当现有看板流程中的流转出现问题时（例如某一列的卡片积压的太多）。因为限制在制品可以更直观地暴露出问题和改进的机会。

如何使用

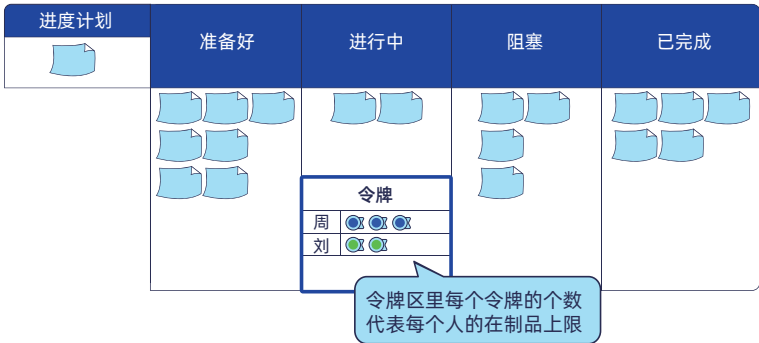
在设置之前...

在设置初始限制在制品数额时不要有太大的压力。初始的限制数额可能无法准确代表您团队的能力，这是完全正常的。在不断的实践过程中，限制在制品的数额也会需要调整从而达到一个新的适合团队的限制。调整的原因可能是：

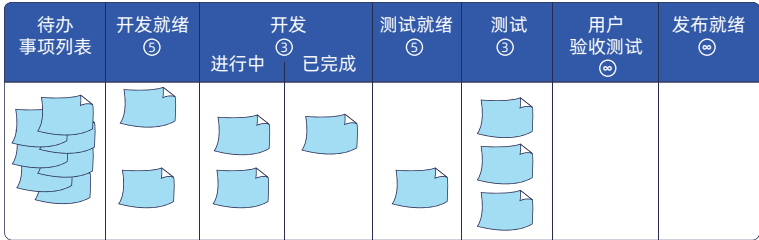
- 工作流发生了变动（例如，增加了新的流程或者移除了某一个状态）；
- 知识、技能和专业度有了提升；
- 团队组成发生了变化（例如，团队成员增加或者离开）但是不论是出于何种原因，定期的评估团队的能力是非常重要的事情。但是当团队达到了限制的数额时，不要着急改变。首先，先讨论一下它发生的原因并且评估一下是否有必要进行调整。如果持续的触及到了限制的数额，那显然团队就需要调整它。并且这也会给出一个信号：流程中有低效率和浪费存在，需要关注它们。

限制在制品没有一套强制的流程规则，以下几种方法可以作为最初起步的一个尝试：

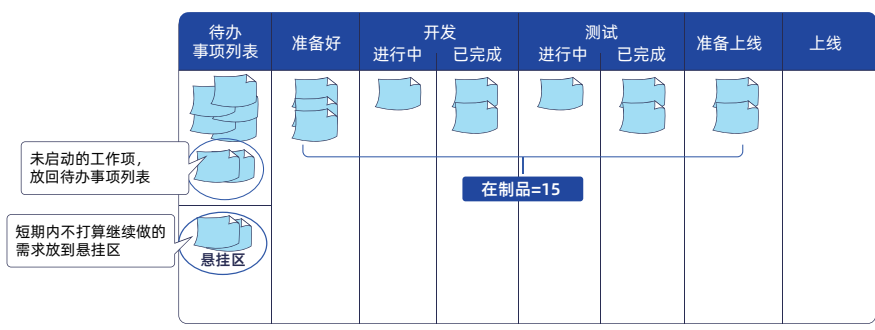
- 按照人数限制在制品，限制人数可以是：
 - 限制在制品数额=人数
 - 限制在制品数额=人数x1.5
 - 限制在制品人数=人数x2
 - 限制在制品人数=人数/2
 - ...



- 按照列限制在制品
 - 根据每一列的情况不同单独设置在制品的数量。初期可以按照处理每种类型的成员的数量决定（例如有3个开发，则开发中就设为3），随着不断的磨合而进行调整达到一个新的平衡值。



- 渐进式方法：用实际的在制品数量作为限制的初始值
 - 步骤1：从看板中去掉以下类型的工作事项：
 - 尚未启动的工作项，将其退出看板放回需求池；
 - 已经启动但是出于搁置状态，短期内又不打算继续做的需求，将他们放置到挂起列中。
 - 步骤2：从看板上剩下工作项的个数作为限制在制品数额的起始值。
 - 按照这个限制运作一个月的时间后，采取渐进式改进的策略尝试降低在制品数量（例如，先尝试降低10%-30%，再运作一个月的时间，回顾会议上讨论是否要继续调整限制）。



- 按照泳道限制在制品
 - 根据每个泳道的不同情况进行在制品数额的限制（如下图所示）

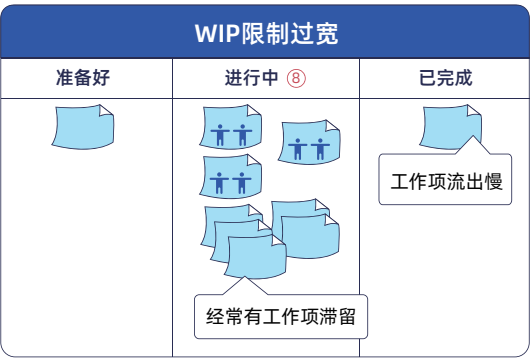
总WIP=20 需求WIP=10 维护任务WIP=3 线上BugWIP=7	就绪	分析 进行中	分析 已完成	开发 准备好	开发 进行中	开发 已完成	构建 准备好	测试	发布 准备好	发布

- 将上述所有方法结合在一起使用，即采用列、泳道或者人数限制结合的方式进行限制。
- 根据团队效率使用（针对整个流程的所有阶段进行限制）
 - 步骤1：绘制你的价值流程图—它就是你流程的各个阶段，代表着看板中的各个列
 - 首先，价值流图的定义是：它是一个过程展示，用于识别和改进生产到最终交付之间成品或服务所需的信息流和物流。
 - 这其中不仅描述了流程的步骤，还描述了每个流程需要多长时间以及它们之间是否有等待时间。这是我们再确定限制在制品时需要特别关注的内容。
 - 步骤2：计算效率
 - 在增加价值上的所花费的时间有多少呢？我们可以使用计算效率的公式：效率=增值的时间/（增值的时间+非增值时间）。

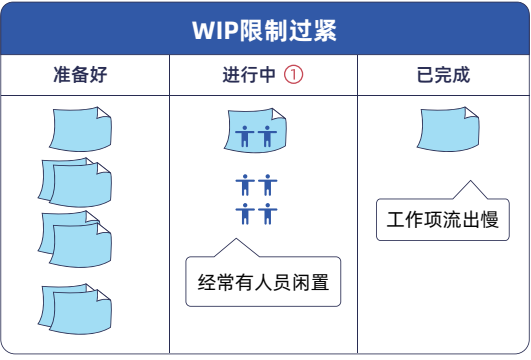
- 步骤3：决定限制在制品数额
 - 为过程的所有阶段确定了一个总限制。下面的这个公式考虑了你在这个项目上投入了多少的人力。限制在制品数额=团队大小（人数）/效率例如，一个团队的增值时间总共花费16天，非增值等待时间消耗9.5天。则整个过程的效率是16/（16+9.5）=63%。若团队大小为5人，则总的限制在制品数额即为：5/63%≈8个工作事件。

如何调整限制在制品数额

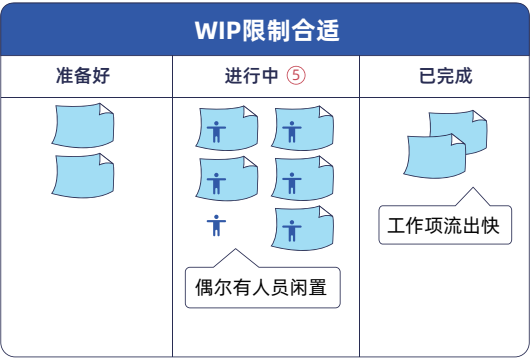
- 如果在制品数额限制过宽，则会经常有工作项滞留，且工作项流出慢。



- 如果在制品数额限制过紧，则会经常有人闲置，工作项的留流出也会变慢。



- 当在制品数额限制到合适的时候，情况会变成：偶尔有人闲置且工作项流出快。所以当团队还未达到这个状态的时候，可以尝试继续调整限制在制品的数额。



项目章程

贡献人：胡俊 晏瑞宇 徐东伟



定义

项目章程，是由项目发起人发布的，正式批准项目成立，并授权项目经理使用组织资源开展项目活动的文件。

主要内容包括：

- 项目背景和愿景；
- 项目目标和关键成果；
- 高层级需求；
- 高层级架构（业务、技术、数据和组织架构）；
- 里程碑计划；
- 预审批的财务资源；
- 关键相关方名单（接口人）；
- 项目审批要求（交付物与验收人）；
- 项目退出标准（关闭或取消项目）；
- 委派的项目经理、核心团队成员及责权；
- 发起人或其他批准项目章程的人员姓名和职权。

实践出处

项目管理知识体系指南，第6版，PMI，4.1-制定项目章程。

为什么

- 在项目与组织的战略目标之间建立连接；
- 在项目团队和需求方之间建立伙伴关系；
- 显示组织对项目的承诺。

何时使用

在项目启动阶段使用

- 与干系人讨论和确认项目目标与范围的讨论材料；
- 作为项目启动会的输入材料。

如何使用

- 项目经理编写初稿；
- 项目团队内部检视；
- 与干系人沟通确认；
- 在项目启动会发布；
- 项目执行过程中持续更新并发布。

项目章程示例：

项目章程

项目名称：_____

项目发起人：_____ 准备日期：_____

项目经理：_____ 项目客户：_____

项目目的或理由：

项目描述：

高水平项目和产品需求：

总预算：

启动风险：

第1页/共3页

项目章程

里程碑总表	到期日

项目目标	成功标准	批准人
范围：		
时间：		
成本：		
质量：		
其他：		

第2页/共3页

项目章程

验收标准：

项目经理职权层级
人员配备决策：

预算管理和偏差：

技术决策：

冲突解决：

超出职权范围的申请途径：

批准：

项目经理签字	发起人或委托人签字
项目经理姓名	发起人或委托人姓名

第3页/共3页

输出物

项目章程

参考资料

- PMBOK没有教你的事——如何写项目章程，简明宏，清晖项目管理培训。
- 图解项目章程应该如何制定及实用模板，PMO前沿。
- 项目启动必备的四张表模板《项目章程》《干系人登记册》《干系人分析矩阵》《干系人管理策略》，PMO前沿。



定义

时间盒是指针对某件事或某个目标，给定一个固定的可用时间，这个时间不能缩小也不能放大。时间盒一般情况下是一个比较短的时间，例如几小时或者几天，人们需要在这个给定的时间内尽全力去达成目标。

实践出处

“时间盒（Timeboxing）”的理念最初从项目管理中衍生而来，本质上讲的是这么一个道理：如果你在规定时间内完不成一项工作，比起花更多时间继续去做这件事，你更应该设法减轻工作量以保证在既定时间内完成。

精益软件开发提出者 Mary Poppendieck 所言：Timebox - don't scopebox。

这是一条极其深刻的智慧，毫不夸张的说，从制造业到互联网行业在内的诸多领域都有它的影子。如我们常听到的精益生产、敏捷开发、产品迭代等等。

为什么

专注：心理学认为安排结束日期为一周之后，比在一个月之后造成的专注效果会更好。时间盒被视为是帕金森定律（Parkinson's Law）的一剂良药：“如何开展工作？只要有效地填满完成前的这段时间。”

提升互信：无论是迭代，还是整个项目，时间盒的另一个价值来自通过频繁达成小的目标来让团队与外部干系人建立信心。

降低复杂度：通过为期两周的时间盒迭代，团队承担的是可管理的复杂度，做他们力所能及的工作。数据表明，低复杂度的步骤能够提高生产率。

更好的过程控制：时间盒是一个以时间盒结束点的决定为标记的短期投资。它使得项目在走得过深而不受控之前，让风险得到更多的最小化机会。

尽早失败或交付价值：由于时间短，它可以更快地反映失败或更早提供价值。更早提供价值，也就意味着生产出了有用的东西。更快的反映失败可以使你更快地知道你是否能达到目标，满足需求，换句话说，如果你不知道何去何从，你可以更快地回到起点，尝试其它的途径。

何时使用

需要获取“为什么”中的各种好处的任何时候都可以使用。

在软件研发中，时间盒常应用在敏捷项目管理中，迭代是时间盒，会议也通常使用时间盒来管理。

如何使用

时间盒的定义可以被总结为以下三步：

计划（Plan）、执行（Do）和复查（Review）

- 1、计划（Plan）：这包括对时间盒的预期产出达成一致，花费多少能达到目标（时间盒的长度、对于技能的要求、预算等），以及度量成功的标准是什么。作为一个指导原则，一般来说，这一步大约需要时间盒的10%-15%。
- 2、执行（Do）：这意味着做实际的工作去完成目标，生产计划中的预期产出。这一步大约点时间盒的70%。
- 3、复查（Review）：这一步包括总结学到了什么，建议或决定下一个时间盒向哪个方向走。

参考资料

- 1、<https://en.wikipedia.org/wiki/Timeboxing>
- 2、<https://www.agilealliance.org/glossary/timebox>
- 3、<https://www.scruminc.com/what-is-timeboxing>



定义

每日站会是一种团队协作的实践。通常是在每个工作日同一时间、同一地点，由所有团队成员参加的会议，时长通常在5-15分钟时间内，部分团队会预先设定站会上需要讨论的议题。这条实践包含以下特点：

- 【每日】意味着它是一种稳定的团队协作行为，固化成了团队共同的行为习惯；同时也意味着这是团队建立的共同规则，团队成员通过遵循这一规则来促进更好的协作。
- 【站】相对于就坐的姿势而言，意味着会议不应该持续太长的时间，提示团队成员应该聚焦重点、简明扼要；同时，站立的方式能让大家保持适度积极的精神状态。
- 【会】意味着这是一种团队成员都应该参与的集中沟通活动，通过此活动来同步一些协作、进展、障碍、目标等等相关的信息。

实践出处

不太清楚最初源自何处，欢迎补充。在Scrum指南中将它作为5个事件之一。

为什么

由于研发工作的复杂性，工作任务的特征、规模和数量等可能会发生各种变化，这些变化可能包括出现的需要掌握的新知、需求理解不一致、需求需要进一步细化、有一些被遗漏的工作、出现技术障碍、有新的替代解决方案、成员之间有相互依赖的工作项、有成员出现了特殊状况等，每日站会致力于通过团队的力量，集中、快速地解决上述问题。

此外：通过每日站会，能解决研发过程不透明的问题，及时充分的把研发真实的进展和状态，以及关键信息提示出来，供团队成员及干系人理解。

通过每日站会上的坦诚沟通与主动协作，能促成团队形成一个真正能够通力合作、紧密联系、协同作战、相互补位、解决难题、产出更优成果的团队。

每日站会通过频繁地开短会、开主题明确的会议的方式，能解决传统会议主题相对发散、沟通效率不高、以及响应速度慢等问题。

总而言之，每日站会的价值包括：

- 对变化做出必要的调整；
- 有助于产生团队认同感；
- 关注并强化集体共同目标与愿景；
- 增强团队士气；
- 减少浪费的时间；
- 有助于找到协作的机会；
- 促进团队的凝聚力；

- 促进知识共享和学习；
- 增强整体的紧迫感；
- 鼓励开发者之间的坦诚和信任关系；
- 营造积极主动的团队文化；
- 改善沟通；
- 发现障碍；
- 促进快速决策；
- 减少不必要的冗长会议等等。

何时使用

团队每天都会召开，通常会约定一个固定的时间，节省了发会议通知的工作量。通常建议上午的早些时刻召开，这样既能给团队成员输入重要的工作目标和内容，也能充分发挥和提升大家的工作积极性（临近下班通常会比较疲惫）。

此外，每日站会对于对进度和响应速度要求比较高的项目尤为适用。也可以用于跨团队协作和外包外协场景。

如何使用

简而言之，就是在固定的时间、固定的地点，所有团队成员一起，沟通一些对团队目标达成有关键作用的重要主题。

这些主题可以参考Scrum 2017版指南中的建议：

- 昨天我做了什么帮助团队达成迭代目标？
- 为了帮助团队实现迭代目标，我今天要做什么？
- 我是否看到任何阻碍我或团队实现目标的障碍？

如果缩减为一个问题则可以是：有什么阻碍我们实现迭代目标？

最新版的Scrum指南删除了这三个问题，指出只要关注迭代目标，并为下一工作日制定可行的计划即可。意味着是否设置主题、以及设置什么主题，可以根据实际场景灵活把握。

看板团队在每日站会上可以关注以下问题：

- 是什么阻碍了我们的工作进展？
- (从看板的最右侧到最左侧)我们取得了哪些进展？

输出物

改变后的任务状态、工作进展（燃尽图）、问题或障碍、协作项、新的工作任务等等。

参考资料

- 1、Scrum指南：<https://scrumguides.org/scrum-guide.html>
- 2、维基百科：https://en.wikipedia.org/wiki/Stand-up_meeting
- 3、模式社区：<https://www.scrumplow.org>

完成的定义

贡献人：薛健



扫码查看详情

定义

用Definition of Done“完成的定义”来表示工作是否已完成，不同的活动有不同的完成定义。

当产品待办列表项或者增量被描述为“完成”的时候，每个人都必须理解“完成”意味着什么。虽然这在不同的 Scrum 团队之间会有巨大的差别，但是团队成员必须对完成工作意味着什么有相同的理解，这样才能保证透明性。这就是 Scrum 团队的“完成”定义，用来评估产品增量是否完成。

实践出处

比较详细的介绍见Mitch Lacey的《敏捷软件开发：Scrum实战指南》（第2版）第七章如何定义“完成”。

为什么

DoD通常会说明：

- 用户故事所处的系统环境（哪个版本的Linux、Android、iOS或者浏览器）；
- 需要输出什么样的文档（自动生成的Javadoc，还是完整的终端用户手册）；
- 有什么质量要求（用于演示的基本功能，还是一个功能完整，健壮的APP）；
- 有什么安全要求（无安全要求，还是从代码评审、代码扫描到网络安全配等各方面都要求做安全审查）；
- 有什么扩展性要求（10个并发，还是10万个并发）。

本质上说，“DoD”是基于验收标准的协议，PO在迭代结束时同它来验收产品增量。

事先确定一个对“完成”的共识，可以为团队与业务节约大量时间来处理反差大、模棱两可或隐藏的工作，通过DoD可以增加价值。

何时使用

使用DoD的时机包括但不限于：

- 迭代0；
- 随着迭代的进展，逐步完善DoD；
- 迭代过程中不同的活动应区别对待，设置不同的DoD。

如何使用

如何制定DoD

DoD明确了团队对于“完成”的预期，可以用来作为迭代Review的检查项，那么如何定义DoD呢？

- DoD应该由团队成员协商得出一致结论；
- 迭代过程中不同的活动应区别对待，设置不同的DoD；

- DoD越弱，欠债越多，后期风险越大。建议对需求流转的各个状态进行DoD设定，会更加严密的保证产品交付质量。

如何制定DoD

- 方式1：召集团队全体成员，讨论各个需求流转状态下的DoD，把最终达成 一致的DoD在团队知识库留存；
- 方式2：让各个流转状态的下游角色为上游角色制定准入条件，比如“开发中-->待测试”的流转DoD，可以由测试同学来写。最后召集团队全体成员，讨论下游角色制定的DoD的合理性，把最终达成一致的DoD在团队知识库留存；
- 方式3：参考下面的DoD表，召集团队全体成员，讨论筛选和增加适合自己团队的DoD，把最终达成一致的DoD在团队知识库留存。

关键状态流传DoD定义参考

状态	DoD说明
需求评审完成	1、全体成员对主流程无疑问； 2、项目管理工具中需求状态变更为评审完成； 3、PM提供埋点文档； 4、需求价值明确，应围绕当月/当季度目标（如提升转化率、提升用户量等）； 5、明确需求数据目标和核心目标的关联计算方式，应同步在项目管理工具中； 6、有完整的需求描述或PRD，并同步在项目管理工具中。
待开发	1、涉及UI的需求，UI图ready，UI、UE和PM对最终设计达成共识； 2、需求已排期，排期信息已更新到项目管理工具中； 3、外部依赖解决，确认外部依赖不会对需求造成阻塞； 4、明确需求上线后的验证方式，如验证数据格式、字段、表结构等，并同步到项目管理工具中。
已提测-待测试	1、前后端接口联调已通过； 2、新增接口文档更新至项目管理工具中； 3、合并代码到上线分支，操作权限仅限xxxx； 4、在测试环境，准入case全部通过（至少提测前一天，QA需准备好准入case）； 5、检查确认无Debug代码，无Debug信息残留； 6、各模块测试用例评审，需在提测前完成，可以和对应开发人员进行。
测试完毕-待验收	1、新增功能测试通过； 2、集成环境中，主流程端到端回归测试通过，无功能回退问题； 3、最高级/严重/主要级别Bug已修复完成且验证通过； 4、低级别遗留Bug，在项目管理工具中标注，下一个上线日前修复； 5、数据埋点测试通过，埋点数据正常； 6、UI和PM走查初验通过，无遗留的阻塞Bug。
验收完毕-待上线	1、UI、PM终验通过； 2、PM对数据埋点验收通过； 3、上线防线、应急回滚方案明确。
已上线	1、上线后，线上功能整体回归通过； 2、上线后，数据埋点回归通过； 3、本次上线的上线记录，同步到项目管理工具上可追溯； 4、记录内容包含不限于:上线操作人、上线日期、上线功能或解决的Bug等。

DoD注意事项

- DoD就是完成准则，DoD就是100%完成，而不是99%、95%、90%的完成。
- DoD必须是团队在项目启动时共同讨论出来的，团队愿意共同遵守的原则，一旦确定，团队就应共同遵守。

- DoD不是一成不变的，在随着时间的推移、经验的积累、成员的变更、项目的变更，我们的DoD也会有很大的不同，所以我們也需要定期地检查和改进。
- DoD越弱，欠债越多，后期风险越大，不同的活动有不同的完成定义，要区别对待。
- 不要把DoD和Accept criteria混淆了，Accept criteria是针对于需求定义的，是我们写在userstory卡片背后的验收标准，AC是站在最终用户的角度，定义产品的外部质量，满足AC确保了我们做正确的事情；而满足DoD确保了我们采用正确的方法做事，它定义的是产品内部质量，保证产品长久的适应性。

DoD分类示例

用户故事DoD

- 用户故事最终的描述符合INVEST；
- 用户故事得到测试用例的对应覆盖；
- 用户故事得到对应的自动化测试用例；
- 用户故事得到用户代表试用并初步认可。

迭代DoD

- 所有完成的用户故事得到PO的验证；
- 所有代码得到静态分析，纠正最高级别的不符合项；
- 所有新增代码得到人工评审；
- 所有完成的用户故事都有对应的测试用例；
- 测试用例都已执行。

发布DoD

- 完成发布规划所要求的重点内容；
- 通过发布的全量测试，回归测试范围是全范围，回归比率不低于50%；
- 修复所有等级为1、2、3的缺陷，4级及4级以下缺陷不超过200个。1、2级缺陷必须修复，3级缺陷经过缺陷发布审批后可以发布；
- 至少通过一次全量回归测试。

版本DoD

- 针对每个版本上线前后的规则；
- 产品文档已全部更新；
- 代码已部署到产品服务器上；
- 运维在验收测试环境上冒烟通过；
- 原始需求提交人对功能已经验收通过；
- 对运维、市场、客服的新功能培训已完成。

每日DoD

- 搭建每日构建环境，晚上自动静态代码检查、编译、部署和测试，每日修复前一日构建和测试发现的缺陷和问题；
- 下班前必须检入当天书写的代码，check in的backlog要填写清晰；

- 当天的代码必须在当天或者第2天邀请同伴进行代码评审；
- 搭建持续集成环境，当天上下午必须至少各检入代码一次（这与第1条可能冲突）；
- 采用TDD，凡是检入的功能代码必须要有对应的单元测试（严格采用TDD）；
- 每天晚上触发静态代码检查、自动化回归测试；
- 当天持续集成、构建环境中的问题，请当天解决。

每日DoD

- 上上周发现的缺陷是否解决；
- 上周新增功能的自动化测试是否加入到每周测试集。

输出物

用户故事DoD、迭代DoD、发布DoD、版本DoD、每日DoD、每周DoD。

定义

就绪定义（Definition of Ready, DoR）是指产品待办事项/用户故事在进入迭代开发之前需要符合的标准。当用户故事符合DoR，即被认为是整个团队充分理解和评估过的，并能立刻开始构建。即“准入”标准。

但请注意，不要把DoR视为冰冷的门槛，而是要通过DoR促进业务与研发的沟通对齐，从而保证产品真正满足业务需要，提升产品质量。

实践出处

DoR通常在Scrum流程中使用，作为用户故事能够进入Sprint的指导意见；

2008年 第一次正式说明“就绪定义”（当时叫 ready-ready，就绪的就绪）；

<https://scrumftw.blogspot.com/2008/10/ready-ready-definition-of-ready-for.html>；

Scrum指南中也有关于就绪的描述，但没有作为Scrum指南的术语；

Scrum指南推荐用Backlog Refinement这个活动来促进业务和开发的沟通并对需求理解一致，从而“就绪”）。

为什么

DoR可以有效防止“低质量需求”流入研发侧，产生浪费。例如：需求描述不清晰、需求价值不明确、外部依赖没有提前识别，故事过大不确定的风险高，故事过大无法在一个迭代完成，无容易测试验收等情况。

通过DoR，可以促进业务和开发之间的沟通和对齐，减少歧义、误解和误会。

通过DoR，可以保证产品符合用户的真实需求，从而保证产品质量。

何时使用

在平日的Backlog refinement活动中，业务和开发一起沟通，梳理需求的细节以符合DoR。

最迟在计划会结束时，用户故事符合DoR，进入当次迭代。

如何使用

自管理团队通过团队公约进行必要的流程和约束，DoR就是其中一项公约。

DoR需要由敏捷团队共同讨论并达成共识，而不是领导或干系人自行决定。如果团队成员不认可或不理解，被动执行的效果大打折扣，敏捷团队的主动性和自组织能力也会降低。

每个团队的DoR可能有所不同。

DoR不是一步到位的，是在迭代中根据团队能力经验和项目阶段进行逐渐完善。

经常会在回顾会中找到的改善点，从而改进DoR。

以下是DoR的一个例子（团队可以自行开发适合于自己的DoR）：

/	条目
1	用户故事要满足INVEST原则
2	验收标准明确
3	提供了原型图
4	用户故事进行了估算并能够在一周内完成
5	风险和依赖已经识别并有应对措施
6	已确认软件架构能够满足要求

输出物

一份经过团队讨论和共识的就绪定义DoR。

常见问题解答

1、DoR和DoD的关系

- DoR是“就绪定义”，表示“该需求已就绪，可以进入迭代”。DoD是“完成定义”，表示“该需求已完成，可以交付使用”。
- DoR有助于输出高质量的用户故事，让开发过程更加高效和准确，符合用户真实需求，保证产品的外部质量。而DoD有助于输出高质量的代码，从而保证产品的内部质量。

2、DoR和AC的关系：

- DoR是团队对需求就绪定义的一致标准。
- AC是针对每个需求的，是这条需求具体需要符合的验收标准。每个用户故事的AC都是独特的。
- DoR中包含“明确的AC”。

3、为什么Scrum指南中没有DoR？

DoR虽然可以帮助提高产品代办事项的质量，但它经常成为开发团队和产品负责人之间的“契约”，成为顺序执行的门槛：只有符合DoR整个检查表的PBI才会被选择用于Sprint Backlog，会增加不必要的过程开销。

Scrum指南为“就绪定义DoR”提供了另一个方案：Backlog refinement 待办事项精化，作为一项持续和频繁的活动来澄清和改进产品待办事项，提高其质量并就绪于下个Sprint的开发。

请参考：
<https://www.scrum.org/resources/blog/why-isnt-defintion-ready-described-scrum-guide>

参考资料

- 1、<https://scrumguides.org>
- 2、<http://scrumbook.org.datasenter.no/value-stream/product-backlog/definition-of-ready.html>
- 3、<https://www.scrum.org/resources/blog/why-isnt-definition-ready-described-scrum-guide>

实战案例

此处仅以《京东敏捷实践指南》为例。每个优秀敏捷团队的DoR都有着自己的成功故事，期待你自己的成功案例！

京东ME团队的DoR：

- 用户故事、业务逻辑、验收标准清晰；
- 有优先级排序；
- 第三方依赖时间确定；
- 有业务期望上线时间；
- 有干系人；
- 交互、视觉、切图完成。

不劳而获，变更免费



扫码查看详情

贡献人：徐东伟

定义

“不劳而获，变更免费”这种敏捷合同是维护一个大合同，同时：

- 1、客户全程积极参与到动态工作范围调整和优先级排序中，确保供应商始终在做优先级最高的事情，因此变更是正常的工作方式，不用额外付款。
- 2、客户要定期给供应商支付已发生的款项，同时，客户可以在任何时候都取消剩余的工作，从而终止项目。客户要为未完成的工作支付 20% 款项。从表面上看，这20%的款项是“不劳而获”的。

为什么

这种机制和激励措施能够让客户提高参与度，同时更早完成工作，节省资金。同样的是，供应商也有激励能够接受变化，把工作做好，得到免费的报酬。

如何使用

Scrum的创始人Jeff Sutherland对“不劳而获，变更免费”进行了详细阐述。这种合同没有采取迷你项目框架的形式，而是维护一个大合同，也就是说要完成一些概括的前期需求分析。但是，合同中加入了两个额外条款。

合同中第一个变化是：要确保总是在开发优先级最高的工作，而且允许加入新工作。客户同意定期与供应商碰面，调整剩余工作的优先级。这时，他们可以向Backlog中加入更多工作，同时要明白：这么说的话，有些工作就得丢掉，再也不会完成了。这有助于增加工作激励，对供应商有帮助，还能保证客户的参与。

与滚动合同类似，客户定期付款，可以按月支付，付款的增量对应交付出来的可工作的软件。这么做保证客户的参与，同时为将来的再次承诺奠定基础，并给后续的改变铺平道路。

“不劳而获”条款让客户在任何阶段都可以取消剩余的工作，并保留目前已经完成的工作。在这种情况下，客户要为未完成的工作支付20%款项。

假如有一个1200万美元、为期12个月的合同已经完成了一半，这时客户希望取消剩余的工作，除了必须要付的600万美元之外，他们还要多付出120万美元。这样相对于整个合同金额，客户还能省下480万美元。

初看上去，供应商可能不会喜欢这种方式，因为这样会让他们少收原来可能赚到的480万美元。然而，他们收到的120万美元却可算是无偿得到的，这应该足以缓解他们的顾虑。假定他们能够马上把人员分配到其他工作上，那么这120万美元就可以算是纯利润了。

目前，这种类型的合同在业界出现得还不多。

输出物

敏捷合同

参考资料

<https://www.infoq.cn/article/agile-contracts>

滚动合同



贡献人：徐东伟

定义

滚动合同是客户与供应商之间达成一个框架协议，通过一系列只需短期开发的小项目来达成目标，在每个小项目完成之后，决定是否继续下一个小项目，形成了滚动效果。

为什么

- 1、保持客户的参与度。
- 2、可以在一个小项目结束时来决定是否继续下一个小项目，如果继续投资的价值已经不大，就可以马上停止，可以让整体投资回报达到最大。
- 3、可以在一个小项目结束时，调整下个小项目的工作范围，优化价值。
- 4、这种合同对供应商也是一个约束，如果供应商在上一个小项目中表现不佳，那么很有可能无法开始下一个项目。

如何使用

合同也许有某些整体目标，但是不需要列出特定的功能特性。“发现需求”是工作的一部分。

伴随着每次交付，客户会付款给供应商，同时面临一个选择：继续下一个小项目，还是就此停止。如果客户认为产生的价值还不如耗费的成本多，他们就可以在已完成工作的基础上就此停手。与之类似，如果供应商发现客户合作程度不够，供应商也可以就此罢手。

某种程度上，这与传统的实践有类似之处，都是要在每个独特的阶段后停下来付款，双方重新承诺，这些阶段包括：需求收集、设计、开发、测试和部署。区别在于：传统模式中，在结束前离开，不会交付任何收益；在这种模式中，总是可以交付一些东西。

从供应商的角度来看：使用已完成功能的纵向切片来展示价值，这能起到明确的激励作用。这是一个常见的敏捷实践。

因为客户能够看到正在完成的工作、正在产生的价值和逐步显现出来的解决方案，他们应该对工作的继续推进充满热情。他们也知道：如果没能达成自己的承诺，他们可以选择离开。这也让他们有信心继续。

这种方式将法律框架从“提供东西”转移到了“提供服务”上。客户签订的是服务合同，而且可以思考他们购买的服务的数量，是4个人月，还是200个工作点数？

虽然这种方式听起来很激进，很多IT部门和供应商已经在相关领域使用服务合同了。举个例子，IT支持部门和软件维护合同一般都是采取服务合同的形式编写。

输出物

敏捷合同

参考资料

<https://www.infoq.cn/article/agile-contracts>

干系人登记册



扫码查看详情

贡献人：胡俊 晏瑞宇 徐东伟

定义

干系人登记册是记录干系人联系信息和分类信息的列表。

实践出处

项目管理知识体系指南，第6版，PMI，13.1.3.1 相关方登记册。

为什么

记录识别到的干系人联系信息，便于沟通时查询，高效沟通；

记录干系人的分类信息，便于引导不同干系人参与项目。

如何使用

一、基本信息

干系人登记册的基本信息包含以下字段：

- 姓名；
- 角色：在项目中承担的角色，例如产品经理、前端开发、后端开发等，根据项目需要细分；
- 联系方式：电子邮件、电话、即时通讯等；
- 位置：对于跨地域项目团队，需要区分地点；
- 职责：负责的业务领域或职能部门；
- 升级路径：如沟通遇到障碍，寻求升级支持的路径。

二、扩展字段

根据引导干系人参与的需要，项目经理可维护以下字段：

- 组织、职位；
- 支持级别；
- 影响力（权力）；
- 关注度（兴趣）；
- 期待；
- 沟通策略；
- 其他字段。

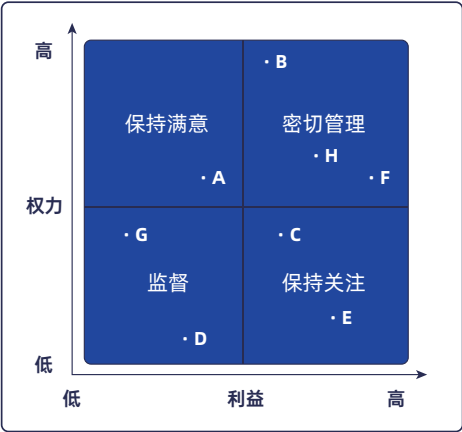
参考字段如下：

姓名	组织职位	项目集角色	支持级别	影响力	沟通	其他特征
相关方1	主管	供应商	中立	低	月度电子邮件	利益
相关方2	客户	接收方	支持	中	周会	需求
相关方3	高级副总裁	发起人	领导	高	季度状态报告	参与状态

注意：干系人登记册可能包含政治和法律上的敏感信息，项目经理可对信息的访问和审查权限设置限制，另存并发布过滤敏感字段后的可公开版本。

三、分析矩阵

项目经理可在干系人登记册新建一个Sheet页，根据相关方的权限级别（“权力”）和他们对项目成果的关注程度（“兴趣”）对相关方进行分组，形成干系人分析矩阵。



输出物

干系人登记册

参考资料

- 1、项目集管理标准（第四版），PMI 表5-1 相关方登记册
- 2、项目管理知识体系指南，第6版，PMI，13.1.3.1 相关方登记册

团队管理

同地团队

贡献人：王用涛 康连波 徐东伟



扫码查看详情

定义

同地团队（Colocated Team）指的是工作在同一个项目上的职能团队共享相同的工作空间，即在同一个地方办公。

敏捷提倡同地团队，团队工作在同一个房间内，桌椅进行合理的摆放，这样大家在视觉、听觉、沟通上的障碍就会变少。并不推荐有墙、隔离区、桌档进行隔挡，因为这样会使得大家感到这是个私人区域，不愿意互相沟通。

为什么

同地团队可以：

- 易于提高面对面交流的速度和效果，特别是通过行为、姿态、语调、面部表情等。
- 易于使用简单的工具进行规划和沟通 ——如白板和便利贴。
- 易于立即澄清问题，很容易可以知道其他成员正在做什么。
- 易于提升信任并且加深共同的尊重。
- 易于简化产品开发。
- 易于节省管理成本。
- 易于团队管理比较容易。
- 易于支持其他团队成员完成任务，因为延迟时间减少，减少导致缺陷或浪费工作的误解，从而提高效率，减少成本。

何时使用

如果在当地有足够的人才和资源，应优先建立同地团队。

团队成员在项目合作时他们在办公区间的位置通常会被保留。而由于敏捷方法很大程度上依赖于短而频繁的反馈循环，因此团队成员发现自己越来越多的出现在会议地点而不是自己的工作区域。敏捷鼓励更频繁地沟通并能适应频繁的改变。

如何使用

- 创建一个既可以开放式交流，又可以深入工作的团队工作区。
- 使用可视化和动手的方法来共同创造和思考。
- 确保你的团队规模是合适的。
- 团队建设非常重要，即使是同一地点的团队也是如此。
- 同地团队也需要有相同的愿景和目标。

- 高度重视“为什么”要在同地办公，要解决什么问题，不能为了在同地办公而在同地办公。
- 同地团队也需要电子系统的支持进行异步沟通，减少沟通成本，从而留出更多时间进行深入的工作。
- 领导要明白，并不是大家坐在一起就自然获得同地办公的收益，领导要以各种形式支持到同地办公团队，如支持团建、支持大家共创等。

参考资料

- 1、<https://blog.csdn.net/chktsang/article/details/86774848>
- 2、<https://statusero.com/blog/how-and-why-a-co-located-team-can-work>

跨职能团队

贡献人：苗再青 晏瑞宇 郑乔尹 徐东伟



扫码查看详情

定义

根据维基百科的说法，跨职能团队是一群具有不同职能专长的人，致力于实现共同目标。提高团队质量的最佳方法之一就是使其具有交叉功能。跨职能团队拥有将想法转变为工作产品的所有必要技能。

Scrum指南表示：Scrum团队由产品负责人、开发人员和Scrum Master组成。Scrum团队是自组织和跨职能的。与组件团队方法相比，跨职能团队是由来自公司不同职能领域的人员组成的团队。它不仅由技术专家（后端、前端开发人员、QA工程师等）组成，还应包括业务分析师、市场营销和用户体验专家或其他积极参与项目的人员。

实践出处

自20世纪80年代开始，跨职能团队被应用于包括丰田、尼桑、本田、宝马、通用汽车、福特在内的大型企业中。团队由来自不同部门、不同层级，甚至不同组织的员工组成，目标是在短时间内完成重要的工作项目，例如新产品开发，提升产品或服务的质量，提升组织效率。

1986年两位学者Takeuchi和Nonaka在“哈佛商业评论”发表的论文《新产品新开发游戏》中也介绍了“异花传粉”的团队，即由具有不同的专业技能、思维过程和行为模式的成员组成项目团队。例如富士施乐公司fx-3500的多功能团队包括计划、设计、生产、销售、分销和评估部门的成员，有助于信息共享、沟通理解，多角度思考和决策，并激发出更多的主动性。

这篇《新产品新开发游戏》通常被认为是Scrum的起源。
原文链接：<https://hbr.org/1986/01/the-new-new-product-development-game>

为什么

稳定及跨职能团队是帮助Scrum团队取得成功和高效工作的关键因素之一。跨职能团队具有更大的灵活性，能够更快地响应不断变化的需求，并能更好地处理持续的支持和维护。

何时使用

在敏捷实施过程中使用。

如何使用

前几年流行的全栈工程师可以说是跨职能团队的一种理想形式，每个人都可以做前端，后端，测试，UED，十八班兵器样样拿得起放得下。但是理想是丰满的，现实是骨感的，因此Mike Cohn表示，“也许敏捷中最普遍和最持久的神话是，跨职能团队是每个人拥有完成工作所需的所有技能的团队。这根本不是真的.....跨职能团队的成员具有各种技能，但这并不意味着每个成员都具备所有技能。”

不管现实和理想有多大的差距，敏捷团队要尽量往跨职能团队努力，而且要尽量保持团队的稳定，只有这样才能保证团队的性能能够持续提升。下面讲一下在现实的场景中，敏捷团队在达不到完全跨职能的情况下，如何运作才能保证敏捷转型能够坚持进行下去。

1、UED是共享团队

即使在大的互联网企业，团队中也很少有配备独立的UED人员，往往是一个大部门有一个UED小组，承接各个开发团队的UED需求。

- 对于这种情况，敏捷团队需要提前和UED团队提出需求，双方定好计划，以便双方的开发节奏一致。刷脸、施展个人魅力、死缠烂打往往是比较实用的招数。有人可能会说，不是应该制定严格的规则，进行优先级排序吗？说这话的人通常是没有经受过社会的毒打。
- 有的同学可能会说，敏捷团队可以自己先按原型进行开发，等UED出来后再作替换。由于UED团队往往会对原型作较大的改动，因此这种方案会造成后期前端工作有较多返工，因此在现实中一般不会被采用，前端同学宁肯等UED的效果图出来后再进行前端工作的开发。
- 还有一种方案是，UED团队先出一版草稿，即只将样式和交互定下来，供敏捷团队开发使用，像颜色、位置、大小等细节后面再深加工。这个方案很符合敏捷原则，但在现实中往往不会被UED团队所采纳，因为这会增加UED团队的工作量。有人可能会说，同一个部门的各个子部门之间不是要互帮互助吗？只能说这是大家共同努力的方向，但是很少有公司或部门能达到这种完美协同的状态。没有这么好的协作，就不进行敏捷转型了吗？敏捷转型就是在挫折和妥协中不断前行。
- 在实践中有一种方法是可行的，我在带团队的时候也采用过。即培养团队内的UED设计能力。但是这并不是要代替UED团队，而是进行有效补充。比如说对于一些简单的确定的小页面，由内部UED人员完成，而复杂的UED设计工作仍采用1)的方法交由UED团队完成。UED设计是有很多创意在里面的，不是一个前端或后端开发人员所有取代的。对于这种情况，UED的标准化工作特别重要，如如按钮大小、页面字体、颜色、对齐等等。
- UED设计，当前流行的工具是蓝湖，产品经理做原型设计的工具一般用Axure。

2、团队中前端人员不足

在敏捷团队中虽然常常没有UED人员，但是前端设计人员确是会配备的。现在都是采用前后端分离的方式，前端只负责显示和页面交互，而业务逻辑基本上都由后端实现，前端通过接口从后端取数。

在实际工作中，后端人员一般能够保证，但是前端人员往往不足，这样便面临着一个问题：一个迭代内的用户故事，后端同学的开发工作完成了，而前端还没有完成，存在后端等前端的情况。这时怎么办？难道后端同学空闲下来等着前端同学？这是部门领导或项目经理绝对不会允许的，不管敏捷教练或管理大师有多好的理由。对于这种情况，在实际工作中有以下方法可以参考：

- 将部分用户故事进行横向切分，即分为两个子用户故事，一个前端用户故事，一个是后端用户故事。比如说一个迭代内规划要完成10个用户故事，当前的前端资源在本迭代内能够完成6个，而剩下的4个按上面所说有方案进行横向切分。当然这种切分方法是完全违背敏捷中的用户故事切分原则的，在实际工作中也是不得已而为之。
- 在团队内培养前端开发能力，即将部分人员往全栈方向发展，既会前端也会后端，这个是长久之计。一开始的时候可以制定一个培养计划，先从简单页面开始，将全栈能力逐渐建立起来。当然这个有一个前提，一是你所在的部门比较稳定，预计缺期内不会被公司裁掉。二是团队预计相对稳定，比如说半年到一年内不会被解散。
- 在大部门范围内协调前端资源：这一点听起来很好，但是在现实中执行起来很难，一是其它团队的前端资源往往也没有整块的时间可以投入到你的团队，只有一些零散的时间，二是临时加入的前端资源还要熟悉代码和业务，往往即使借调过来也很难起到很大作用。但是这不失为一个思路，可以尝试，在实际工作中大家也在这么做。
- 将尽可能多的工作往后端移，使前端尽可能的只是纯做展现工作，这样会减少前端工作量，缓解前端瓶颈。但是有时候会遇到性能问题，如果前端页面对于性能要求较高，有些显示逻辑在前端实现往往效率会更高。如何权衡，需要根据具体情况而定。
- 将前端同学的“杂事”尽可能分解出去，由别人承担。

学习工作坊

3、团队中的各个成员只熟悉产品中的一两个模块

团队中各个成员只熟悉产品中的一两个模块，这个在大多数团队中都存在，造成这种情况最主要原因是团队迫于交付压力，往往会将需求交给最熟悉的人员去开发，这样效率无疑是最高的。

这个问题不解决，会造成敏捷团队内各人的工作不均衡，严重影响团队效率的提升，也会影响团队的工作氛围。在实际工作中可以采用以下方法：

- 还是往全栈方向发展，这个全栈不是指工作类型的全栈，而是指业务能力的全栈。一种方法是将产品的每个模块都设置A/B角，A,B两个人对这个模块都很熟悉，另一种方法是让团队中每个人对产品中的大部分模块都熟悉，做到一精多熟，即精通一个模块，对于其它模块也熟悉，也能承接开发工作。这个需要在团队内制定详细的能力提升计划，并在工作中安排学习时间，我在实际工作中往往在团队内每周安排一个小时进行业务和技术培训。
- 可以先从简单的需求开始，当然在安排工作时，要多给出一部分学习时间。例如张三同学熟悉A模块，现在有一个用户故事是B模块的，安排张三同学去做，如果正常时间是12个小时，则这次给张三同学16个小时，其中四个小时是学习时间。同时要给张三同学安排明确的导师，在开发中给予支持。
- 在实际工作中，我会在迭代计划中，让SM列出各个人在本迭代的工作量。对于工作量不饱和的同学，在会上引导说：“Scrum五大价值观中，有一个是勇气。现在希望有同学勇敢地挑战一下自己不熟悉模块的用户故事。”这时，往往会有人主动站出来，这个方法屡试不爽，可见每个人都有一颗骚动的心，只等着你去挑动。

4、敏捷团队成员经常变动

出现这种情况往往有两个原因：

- 团队成员中很多是外包人员，这个在制造业的IT部门存在较多。外包人员往往是按项目外包的，项目完成后则人员离场。对于这种情况，需要组织进行变革，将项目外包转为长包（象一些互联网企业，一些国企，一般采用年包制），这个变革是艰难的，需要敏捷转型人员付出持续艰苦的努力。如果组织短时间内无法进行变革，是需要将多个项目的外包拼接起来，使外包人员做完一个项目，然后进入另一个项目，形成一种事实上的长包，但这会花费产品经理很大精力。
- 团队是按项目组织的，项目结束后，团队人员就解散了。在实际工作中，敏捷教练需要推动组织由项目团队往特性团队转变，即敏捷团队是相对稳定的，可以承接来自各个方面的需求，这些需求有可能是来自于多个项目的，也可能是一些维护性小需求。对于团队而言，它不需要知道需求是从哪里来的，它所面对的是一个一个的用户故事。当然由于团队可能同时在干两个项目的需求，还有一些线上Bug的修复以及维护性的小需求等，需要进行严格的代码分支管理，在此不展开讲，大家可以看一下代码分支方面的实践分享。

输出物

面对团队当前的问题而采取的改进措施。

常见问题解答

- 1、因为团队不具备跨职能特性，于是认为当前团队不具备敏捷转型的条件，又回到传统的瀑布模式；
- 2、由于团队资源不足，认为领导对敏捷不了解不支持，心存报怨，而不是面对问题解决问题。

敏捷工作坊

敏捷多米诺骨牌工作坊

贡献人：徐东伟 Eason



定义

该工作坊使用多米诺骨牌为道具，让大家能够在游戏当中领会Scrum的运作方式，体会Scrum给我们带来的好处。

实践出处

该工作坊是由敏捷教练侯伯薇创造，并曾在全国举办了许多场次。

为什么

- 1、让学员了解如何通过相互协作的方式达成团队共同的目标；
- 2、让学员更好掌握Scrum的基本框架，包括价值观、角色、工件和事件；
- 3、让学员了解以Scrum方式运行软件项目的时候，可能遇到的问题，并找到合适的解决方案。

如何使用

人数：20-30左右，分为四组。

时间：2-3小时。

场地：每组要有一张足够大小的桌子，场地要有一些平整的空地，因为有时候可能会在地上摆。

准备材料

- 1、2000块左右多米诺骨牌（材料为塑料和木头均可）；
- 2、四套机关（一般购买骨牌的时候会附带）；
- 3、四套任务卡（见下方资源链接）；
- 4、四套特殊摆法说明（见下方资源链接）；
- 5、四套机关使用说明（见下方资源链接）；
- 6、海报纸数张，彩色水彩笔数支。

扫码获取资源：



过程

- 1、创建任务板，包含三个栏位即可：To do、Doing、Done，用来流转任务卡。
- 2、团队角色划分：PO、SM、Team。
- 3、发放任务卡，并请各组裁剪开，在这个过程中大概了解任务卡的内容。
- 4、开计划会议，请各组澄清需求，并选择将在第一个Spint中要完成的任务卡；团队成员领取任务，在任务卡上签名；把选择出来的任务卡放在任务板的“Todo”栏中。
- 5、开始第一个Sprint，各组开始按照选择的任务卡进行任务，并适时移动任务卡到合适的栏位中。
- 6、如有必要，在中间插入Daily Scrum，请组员用三个问题的形式陈述自己的进展。
- 7、结束第一个Sprint，记录完成的任务的价值总和，带各组总结收获，并制定下一轮的改进策略。
- 8、重复4-7步骤三次，相当于完成三个Sprint。
- 9、做最终的回顾。

可能的学习点

- 1、整个过程是一个不断迭代、不断学习、不断改进的过程，正符合敏捷“持续改进”的理念。
- 2、团队中成员所擅长的方面各不相同（有些人擅长比较简单而风险低的，有些人擅长难一些、分享高而价值高的，有些人擅长做协调和管理的工作等等），应该让合适的人做合适的事情。
- 3、骨牌的排放可以借助工具，但开始不给，扮演用户的引导师可以在手上拿着。这样可以让大家知道沟通的重要性，以及工具能带来的效率提升。
- 4、各组因为成员不同导致策略也会有所不同，有些组会更愿意做高难度的任务，有些组更保守一些，稳扎稳打，也有些组会搭配组合，达到最好的效果。
- 5、在进入一个不熟悉的领域时，需要一个循序渐进的过程。
- 6、有时候会高估任务的难度，也有时候会低估任务的难度。
- 7、成员相互之间可能会存在依赖和干扰。
- 8、工作的环境很重要，可能会对工作产生比较大的影响。
- 9、时间分配要合理，争取在每个sprint中都有产出。
- 10、有些高难度的工作，可以进一步拆分，多人配合完成。
- 11、做技术的同学，有可能更关注过程，却忽视了结果的重要性。
- 12、做任何任务，都要考虑投入产出比。

致谢

自2022年7月1日发布0.1版本至今，RRPL受到了众多小伙伴们的关注，也有很多小伙伴为RRPL贡献了自己的力量，如对已有的RRPL实践做出修改、从0到1创建新实践等等。在此，我们要感谢如下共创者对RRPL做出的贡献：

Becca、曹江辉、柴颖、陈正思、程家乐、董越、杜春、Eason、龚敏睿、顾伟峰、何凡、胡俊、荆修梅、巨新龙、康连波、李虎、鲁飞、露西、栾绍晨、苗再青、任甲林、宋晓娜、王磊、王连杰、王婷、王岩、王用涛、王增舟、吴言、徐东伟、许慎、薛健、晏瑞宇、张乐、张鹏、郑乔尹、朱美玲。
(按照首字母排序)

同时，也非常感谢《软件研发效能度量规范》团体标准组委会及各位专家的授权，为RRPL的“度量”实践贡献了力量，将业界智慧的结晶通过RRPL共享给广大小伙伴！

除此之外，我们在筹备这本RRPL手册的同时，也已有更多小伙伴加入了为RRPL贡献实践的过程中来。所以，当大家拿到这本RRPL手册的时候，我们的RRPL已经有了更多新的实践，欢迎大家关注RRPL网站，及时查看最新版本的RRPL~



扫码查看RRPL更多内容

词汇索引表

AC（Actual Cost）：实际成本 /218

也称ACWP（Actual Cost for Work Performed，已完成工作量的实际费用），指项目实施过程中某阶段实际完成的工作量所消耗的工时或费用。

AC（Accept Criteria）：验收标准 /102

在验收项目可交付成果之前必须满足的标准，包括绩效要求和基本条件。

Agile Development：敏捷开发 /096

指具备如下特征的软件开发方法：开发人员与客户持续协作、以用户故事及相应验收测试形式存在的有限需求文档以及快速而频繁交付有用功能的小增量。敏捷开发方法包括极限编程Scrum、特性驱动开发、精益看板方法等。

Agile Domino Workshop：敏捷多米诺骨牌工作坊 /250

工作坊使用多米诺骨牌为道具，让大家能够在游戏当中领会Scrum的运作方式，体会Scrum给我们带来的好处。

ATDD（Acceptance Test Driven Development）：验收测试驱动开发 /016

ATDD是一种开发方法论。在ATDD活动中，团队需要就需求定义出期望的质量标准和验收细则，以明确而且达成共识的验收测试计划（包含一系列测试场景）来驱动产品的代码开发和测试脚本开发。

Backlog：得办事项列表细化 /232

在敏捷项目中，项目尚未开始的工作经过优先级排序所形成的有序列表。一个Backlog中可以包含用户故事、业务过程、变更请求、基础设施开发以及缺陷故事。Backlog中的工作事项基于其优先顺序分配给接下来的各个迭代。

Backlog Refinement：待办列表细化 /234

向待办事项中的项目添加细节、估计的行为，这是一个持续的过程。

Baseline：基线 /048

记录当前已经达成共识、经过评审并审核通过的一系列需求的即时快照，通常定义某一具体产品版本或开发迭代的内容。基线是未来开发工作的基础。

BDD（Behavior Driven Development）：行为驱动开发 /064

是敏捷领域中测试驱动开发（TDD）的一种延伸技术，它旨在鼓励软件项目中涉及到的人员角色（包括开发者、QA和非技术人员或商业参与者等）之间的协作。

Burndown Chart：燃尽图 /229

用于表示剩余工作量的工作图表，由横轴（X）和纵轴（Y）组成，横轴表示时间，纵轴表示工作量。

BVT（Build Verification Testing）：构建验证测试/版本验证测试 /107

指在一个版本构建完成之后，团队自动运行的一套验证系统的基本功能的测试。

CCB（Change Control Board）：变更控制委员会 /048

是由干系人正式组成，并负责决定接受或否决提出的(包括需求变更在内的)变更的一组人，变更控制委员会的所有决定和建议均应记录在案。

CI（Continuous Integration）：持续集成 /061

持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译、发布、自动化测试）来验证，从而尽早地发现集成错误。

Code Coverage：代码覆盖率 /184

是衡量产品质量的关键指标之一。指软件产品发布后发现的缺陷数量与该软件产品在整个生命周期发现的所有缺陷数量的比率，通常用来衡量软件团队对软件质量控制的水平。

Code Review：代码评审 /061

代码评审也称代码复查，是指通过阅读代码来检查源代码与编码标准的符合性以及代码质量的活动。

Colocated Team：同地团队 /244

指工作在同一个项目上的职能团队共享相同的工作空间，即在同一个地方办公。

Component Testing：组件测试 /163

组件通常指大型系统中任何封装良好、连贯且可独立替换的中间子系统，在微服务架构中，一般代表单个微服务，因而组件测试就是对单个服务的测试。

Condition Coverage：条件覆盖率 /185

指的是在测试时运行被测程序后，所有判断语句中每个条件的可能取值（真值和假值）出现过的比率。

Control of Progress：进度控制 /130

是对生产和工作进度所进行的控制。它是同时时间的进程和期限的需求结合在一起的，主要用于控制生产或工程进度是否按期完成，各项作业在时间上能否相互衔接等。

Cross-functional Teams：跨职能团队 /246

指把各种工作领域具有不同知识、技能的员工组合起来识别和解决共同的问题的团队，致力于实现共同目标。

Daily Build/Nightly Build：每日构建/每夜构建 /086

是每天完成最新版本程序的软件构建的做法。这样可以首先对其进行编译以确保存在所有必需的依赖项，并且可能经过测试以显示没有引入任何错误。每日构建通常也是公开可用的，允许访问最新功能以获取反馈。执行每日构建有助于确保开发人员能够合理地确定出现的任何新错误都是他们自己在最后一天完成的工作的结果。

Decision Coverage：判断覆盖率 /185

是指执行足够的测试用例，使得程序中的每一个分支至少都通过一次。使得每一个判定获得每一种可能的结果。“判断覆盖”比“指令覆盖”严格，因为如果每个分支都执行过了，则每个语句也就执行过了。

Defect：缺陷 /062

项目组成部分中不能满足要求或技术规格，需要修补或更换的瑕疵或缺点。

DEP（Defect Escape Percentage）：缺陷逃逸率 /181

缺陷逃逸率是衡量产品质量的关键指标之一。是指软件产品发布后发现的缺陷数量与该软件产品在整个生命周期发现的所有缺陷数量的比率，通常用来衡量软件团队对软件质量控制的水平。

DevOps /075

DevOps是Development和Operations的组合同，是一种重视软件开发人员（Dev）和IT运维技术人员（Ops）之间沟通合作的文化、运动或惯例。透过自动化软件交付和架构变更的流程，来使得构建、测试、发布软件能够更加地快捷、频繁和可靠。

DoD（Defifinition of Done）：完成的定义 /230

用来表示工作是否已完成，不同的活动有不同的完成定义。

DoR（Defifinition of Ready）：就绪定义 /234

是指产品待办事项/用户故事在进入迭代开发之前需要符合的标准。当用户故事符合DoR，即被认为是整个团队充分理解和评估过的，并能立刻开始构建。

DSDM（Dynamic Systems Development Method）：动态系统开发方法 /032

DSDM是众多敏捷开发方法中的一种，倡导以业务为核心，快速而有效地进行系统开发。

Dynamic Analysis：动态分析 /104

动态分析就是通过观察软件运行时的动作，来提供执行跟踪、时间分析，以及测试覆盖度方面的信息。

ELOC（Equivalent Line of Code）：代码当量 /137

是一种对开发者代码工作量进行合理量化和度量的指标。与代码行数、提交数等浅层工作量指标相比，代码当量的优势体现在两个方面：不易受到编程习惯或特定代码行为的干扰（如换行、空行、注释、括号等），且能更好地反映代码开发所涉及的逻辑量。

Encapsulate Field：封装成员变量 /072

将仅限于本类使用的变量重写成私有成员变量，并提供访问方法。这种方式可以将与外部调用者无关的变量隐藏起来，减少代码的耦合性，减少意外出错的概率。

Entry/Exit Coverage：进入点/结束点覆盖率 /185

是指测试时对函数中所有可能的进入点及结束点的执行比率。

EV（Earned Value）：挣值 /218

也称为BCWP（Budgeted Cost for Work Performed），指项目实施过程中某阶段按实际完成工作量及按预算定额计算出来的费用。

Extract Method：提取方法 /072

大段代码中的一部分提取后，构成一个新方法。这种重构可以使整段程序的结构变得更清晰，从而增加可读性。

Fibonacci Sequence：斐波那契数列 /041

斐波那契数列又称黄金分割数列，因数学家莱昂纳多·斐波那契以兔子繁殖为例子而引入，故又称为"兔子数列"。斐波那契数列由0和1开始，之后的斐波那契数就是由之前的两数相加而得出。首几个斐波那契数是：1、 1、 2、 3、 5、 8、 13、 21、 34、 55、 89、 144、 233、 377、 610、 987.....

特别指出：0不是第一项，而是第零项。

Function Coverage：函数覆盖率 /185

覆盖率就是执行过的代码占总代码的比例，比如执行了多少行（Line），执行了多少个分支（Branch），执行了多少个函数（Function），执行了多少条语句（Statement）。

Function Point: 功能点 /065

功能点是度量软件规模的一个标准度量单元。简单的说，一个软件的大小可以通过交付给用户的功能点数来度量。

Functional Testing: 功能测试 /200

根据产品特性、操作描述和用户场景，测试产品的特征和可操作行为，以确定其满足设计需求。

Generalize Type: 类一般化 /072

将多个类/函数共用的类型抽象出可以公用的基类，然后利用多态性追加每个类/函数需要的特殊函数。可以让结构更加清晰，同时可以增加代码的可维护性。

Hypothesis Testing: 假设检验 /213

又称统计假设检验，是用来判断样本与样本、样本与总体的差异是由抽样误差引起还是本质差别造成的统计推断方法。

ITIL (Information Technology Infrastructure Library)：信息技术基础架构库 /179

主要适用于IT服务管理（ITSM）。ITIL为企业的IT服务管理实践提供了一个客观、严谨、可量化的标准和规范。

ITSM (ITService Management)：IT服务管理 /179

是一套帮助企业对IT系统的规划、研发、实施和运营进行有效管理的方法论。

JJ-Path Coverage: JJ路径覆盖率 /185

是路径覆盖的变种。

Law of Large Numbers: 大数定律 /155

是概率论中讨论随机变量序列的算术平均值向随机变量各数学期望的算术平均值收敛的定律。大数定律分为弱大数定律和强大数定律。

LCSAJ (Linear Code Sequence and Jump): LCSAJ覆盖率 /185

是根据程序本身决定的代码。它的起点可以是程序第一行或转移语句的入口点，或是控制流可跳达的点。在程序中，一个LCSAJ是一组顺序执行的代码，以控制跳转为其结束点。

Lead Time for Changes: 变更前置时间 /152

是指从代码更改提交到主干分支直到它处于可部署状态之间的时长。

Lean Kanban: 精益看板 /205

是一种敏捷和精益开发方法的结合，通过可视化开发来改善协作、优化管理，显著提高交付速度、质量和灵活性。

Limit WIP: 限制在制品 /220

限制工作流每个状态或全流程中可以存在的最大工作量，通常体现在看板上，而它限制的对象可以是个人、团队、甚至是整个组织。

Little's law: 利特尔法则 /206

利特尔法则可用于一个稳定的、非占先式的系统中。在一个稳定的系统（L）中，长期的平均顾客人数，等于长期的有效抵达率（λ），乘以顾客在这个系统中平均的等待时间（W）。可以用一个代数式来表达：L=λW。

Loop Coverage: 循环覆盖率 /185

指的是所有循环是否都有运行过零次、一次及一次以上的测试。

MTBF (Mean Time Between Failure) 平均故障间隔时长 /159

一个设备或系统从发生故障到发生下一个故障的平均正常运行时间。它用于预测系统和组件的可靠性和可用性。通过跟踪系统和组件故障之间的正常运行时间来计算该指标，以便及时维修或者更换系统组件。

MTBSI (Mean Time Between System Incidents)：系统事件间隔平均时间 /180

两次连续事件之间的平均经过时间。一般通过将MTBF和MTRS（MTBSI=MTBF+MTRS）相加来计算。

MTTD (Mean Time To Detect)：平均检测时间 /179

从发生问题到检测到问题的平均时间。MTTD指示的时间段是直到IT团队收到故障单，MTTR的测量从这一时间点开始。

MTTF (Mean Time To Failure)：平均无故障时间 /180

设备或系统出现故障之前的平均预期正常运行时间。IT团队通常需要花费数天或数周的时间观察系统组件以收集这些数据。

MTTI (Mean Time To Investigate)：平均调查时间 /179

从发现IT事件到组织开始调查原因和解决方案之间的平均时间。

MTTR (Mean Time to Recovery 或 Mean Time to Restore)：平均服务恢复时间 /177

指的是从检测到事件发生到目标系统或组件再次可供用户使用的平均经过时间，它是衡量组织的系统、设备、应用程序和基础设施的可维护性以及在进行IT事件时，恢复IT服务效率的关键指标。

MTTR (Mean Time To Repair) 平均故障修复时长 /158

是修复系统故障并将其恢复到完整功能所需的平均时长。指标反映了线上故障恢复能力，通过缺陷快速修复或回滚降低平均故障修复时长是改进的重点之一。

Parameter Value Coverage: 参数值覆盖率 /185

指一个方法的所有参数，是否执行过其中最常见数值。

Path Coverage: 路径覆盖率 /185

是指在测试时运行被测程序后，程序中所有可能的路径被执行过的比率。

Performance Testing: 性能测试 /200

性能测试是指通过测试工具和测试手段，监测和收集测试过程中的软件系统运行数据，度量系统与预定义目标的差距。

PO (Product Owner)：产品负责人 /022

敏捷项目团队中的角色，负责建立产品的愿景，提供项目边界和约束，对产品待办事项列表中的内容排列优先级顺序，制定产品决策。

PRD (Product Requirement Document)：产品需求文档 /015

PRD是产品项目由“概念化”阶段进入到“图纸化”阶段的主要文档，是对产品需求（用户需求、竞品分析、运营团队、战略目标等）进行理解、梳理和定义描述后的产物。

Predicate Coverage: 谓词覆盖率 /185

又称判定覆盖，要求对于一组谓词中，测试需求包括每个谓词的结果为真和为假。

Product: 产品 /003

项目正在开发的任何最终交付物。产品、应用、系统以及解决方案等都统称为产品。

Product Line: 产品线 /076

是指一群相关的产品，这类产品可能功能相似，销售给同一顾客群，经过相同的销售途径或在同一价格范围内。

Product Manager: 产品经理 /004

是指在公司中针对某一项或是某一类的产品进行规划和管理的人员，主要负责产品的研发、制造、营销、渠道等工作。产品经理是很难定义的一个角色，如果非要一句话定义，那么产品经理是为终端用户服务，负责产品整个生命周期的人。

Project: 项目 /002

项目是在规定的时间、预算和质量目标范围内完成的各种工作，属于战略层面的概念。

Project Charter: 项目章程 /224

项目章程是由项目发起人发布的，正式批准项目成立，并授权项目经理使用组织资源开展项目活动的文件。

Pull Up: 函数归父 /072

指的是方法从子类移动到父类。

Push Down: 函数归子 /072

指的是方法从父类移动到子类。

PV (Planned Value)：计划值 /218

也称BCWS（Budgeted Cost for Work Scheduled），指项目实施过程中某阶段计划要求完成的工作量所需的预算费用。

QA (Quality Assurance)：质量保证 /066

质量保证是指为了提供足够的信任表明实体能够满足质量要求，而在质量管理体系中实施并根据需要进行证实的全部有计划和有系统的活动。而担任这类工作的人员就叫做QA人员。

QFD (Quality Function Deployment)：质量功能展开 /038

QFD法通过量化分析顾客需求与工程措施间的关系度，找出对满足顾客需求贡献最大的关键措施，从而指导设计人员抓住主要矛盾，开展稳定性优化设计，开发出满足顾客需求的产品。这是一种旨在提高顾客满意度的“顾客驱动”式的质量管理方法。

RAD (Rapid Application Development)：快速应用程序开发 /032

提倡快速的原型设计和即时反馈，缩短开发时间并加快交付速度，是以最小幅度的规划并迅速地將原型完成的软件方法论。

RCA (Root Cause Analysis)：根因分析 /208

根因分析是一项结构化、系统化的问题处理过程，包括确定和分析问题原因，找出问题解决办法，并制定问题预防措施。

Rename Method: 方法更名 /072

方法的命名应准确地表达其用途。

Requirement: 需求 /002

对用户诉求和目标的陈述，或者产品为满足这样的诉求或目标而必须拥有的条件或能力。产品要向干系人提供价值时必须具备的属性。

Rollback: 回滚 /112

指的是程序或数据处理错误，将程序或数据恢复到上一次正确状态的行为。回滚包括程序回滚和数据回滚等类型。

SAFe (Scaled Agile Framework)：规模化敏捷框架SAFe /040

SAFe是一个规模化敏捷框架，不仅包括团队敏捷，还包括了价值流、投资组合、项目集等层级的敏捷管理方法和架构。

Scrum /006

Scrum是一种迭代式增量软件开发过程，是敏捷开发的重要框架之一，通常用于软件行业，目前也已扩展到其他领域。

Scrum Master /006

Scrum团队的专家，需要通过知识讲解、培训、游戏、沙盘演练等方式传递Scrum知识，帮助团队掌握Scrum的思想和实践，实现Scrum落地。

Security Testing: 安全测试 /200

在软件产品的生命周期中，特别是产品开发基本完成到发布阶段，对产品进行检验以验证产品符合安全需求定义和产品质量标准的过程。

SEI (Software Engineering Institute) 软件工程研究所 /047

是一个软件工程领域的研究机构，它所开发的软件能力成熟度模型（CMM）系列，在全球范围得到普遍的认可和应用。

Shadow Domain Releasing: 影子域发布；

Shadowen-vironment Releasing: 影子环境发布；

Live-live-releasing: 双热发布。 /115

是蓝绿部署的一种变形，将试运行环境和生产环境作为蓝绿环境。将应用程序的新版本部署到试运行环境上，然把用户从生产环境引导至试运行环境中，让用户开始使用这个新版本。此时，试运行环境就变成了生产环境，生产环境就变成了试运行环境。

Shared-nothing Architecture: 非共享架构 /113

是一种分布式计算架构。这种架构中的每一个节点都是独立而且整个系统中没有单点竞争。有些系统需要集中保存大量的状态信息，如数据库、应用服务器或是其他类似的单点竞争系统。

SIT(System Integration Testing): 系统集成测试，也叫做集成测试 /099

系统集成测试是一种黑盒测试技术，它可以根据特定的需求来评估系统的遵从性。

SLA (Service-Level Agreement)：服务等级协议 /129

是协议双方（服务提供者和用户）签订的一个合约或协议，这个合约规范了双方的商务关系或部分商务关系。一般来说，SLA是服务提供者与用户之间协商并签订的一个具有法律约束力的合同，合同规定了在服务提供过程中双方所承担的商务条款。

SMART：SMART原则 /021

SMART原则是一种目标管理方法，包括五项原则，分别是有时限的（Time-bound）、具体的（Specific）、有相关性的（Relevant）、现实可达到的（Attainable）可衡量的（Measurable）。

SME（Subject Matter Expert）：主题领域专家 /048

指对某个领域有丰富经验和知识的个人，通常被人们视为该领域相关信息的权威来源。

Smoke Testing：冒烟测试 /107

冒烟测试是对产品版本的基本功能进行确认验证，以及确定和修复缺陷的最经济有效的方法。能快速验证产品版本的基本功能是否已经实现，是否会破坏产品版本的稳定性，从而确定能否继续进行更多的测试。

Sprint /004

也称为冲刺或迭代，指Scrum团队完成一定数量工作所需的短暂、固定的周期。

Stakeholder：利益相关者/干系人 /006

参与项目并受过程或结果所影响，或能够影响过程或结果的个人、团队或组织。

Statement Coverage：指令覆盖率 /185

是一个比较弱的测试标准，是指选择足够的测试用例，使得程序中每个指令至少都能被执行一次。

Static Analysis：静态分析 /075

静态分析就是对软件的源代码进行研读，查找错误或收集一些度量数据，并不需要对代码进行编译和执行。

Story Point：故事点 /025

故事点是一个度量单位，用于表示完成一个产品待办项或者其他任何某项工作所需的所有工作量的估算结果。

System Testing：系统测试 /099

对整个系统的测试，将硬件、软件、操作人员看作一个整体，检验它是否有不符合系统需求的地方。

TDD（Test-Driven Developmen）：测试驱动开发 /064

是敏捷开发中的一项核心实践和技术，也是一种设计方法论。TDD的原理是在开发功能代码之前，先编写单元测试用例代码，测试代码确定需要编写什么产品代码。

Timeboxing：时间盒 /032

指针对某件事或某个目标，给定一个固定的可用时间，这个时间不能缩小也不能放大。

UED（User Experience Design）：用户体验设计 /247

是以用户为中心的一种设计手段，以用户需求为目标而进行的设计。

Unit Testing：单元测试 /104

单元测试是用来对一个模块、一个函数或者一个类来进行正确性检验的测试工作。

Value Flow：价值流 /147

通常是指生产制造过程中，从原材料到成品、并赋予其价值的所有活动。

Weibull Distribution：韦伯分布 /151

又称韦氏分布或威布尔分布，是可靠性分析和寿命检验的理论基础。在可靠性工程中被广泛应用，尤其适用于机电类产品的磨损累计失效的分布形式。由于它可以利用概率值很容易地推断出它的分布参数，被广泛应用于各种寿命试验的数据处理。

Wide band Delphi：宽带德尔菲法 /044

宽带是由PERT和德尔菲法相结合得到的方法，主要思路是德尔菲法中参与估计的群体需要进行估计的是三种时间：最乐观时间、最保守时间和最可能时间。德尔菲法是组织专家就某一专题达成一致意见的一种信息收集技术。相关专家匿名参与。组织者使用调查问卷就一个重要项目事项征询意见，然后对专家的答卷进行归纳，并把结果发还给专家，请他们做进一步评论。这个过程重复几轮后，就可能取得一致意见。德尔菲法有助于减轻数据的偏倚，防止任何个人对结果产生不恰当的影响。

WIP（Work In Progress）：在制品 /152

指还处在生产过程中但尚未完成的产品/工作。

XP（Extreme Programming）：极限编程 /024

极限编程是由Kent Beck在1996年提出的一种软件工程方法学，是敏捷软件开发的一种方式。极限编程和传统方法学的本质不同在于它更强调可适应性性能性以及面临的困难，适用于小团队开发。