敏捷软件估算和度量

软件估算&度量的意义

人是无法管理自己不了解的事物。如果不看天气预报上的天气信息，我们无法决定是否带伞。如果不做远途行程的时间计划和路径计划，我们就不知道和朋友的约会应该什么时候出门，选择什么交通方式最合适。这就是估算和度量最基本的作用，能帮助信息获得者做出正确的活动决定。

同样在软件行业，软件度量的根本目的是为了管理的需要，利用度量结果来改进软件研发过程和产品方向。因为研发过程对于管理者来说是个黑盒，他们不一定也不需要了解研发过程的每个细节，但他们需要在宏观上掌握研发进展状态，他们需要为研发的成本，进展，风险，战略方向等基于准确的数据结果做出决策，不然就是我们通常称为拍脑袋决定方向的指挥者。

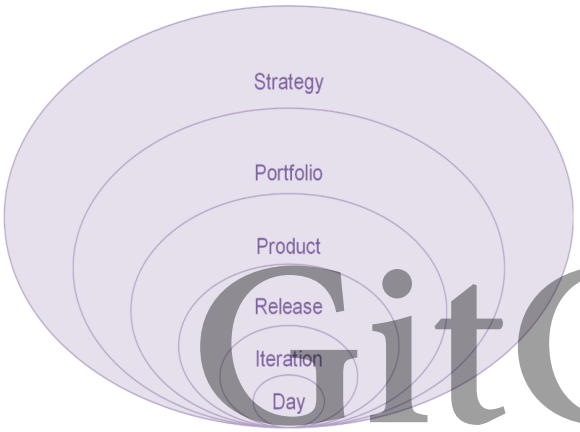
软件度量最基本的估算和度量范畴是：**进度、成本、质量**。不管采用何种研发方式，这三项指标是最根本的，其它度量项是在此基础上进行的拓展和深挖，如质量可以包含单元测试覆盖率是质量的深挖，又如效率=规模*进度/成本效率，可作为绩效考核的一个指标。把握了这三项，管理的度量维度才算是完整的，这就能解答我们一系列的问题：**项目计划，成本控制，项目监控，量化管理。**

甚至，如果有外包供应商，那么他们的绩效如何，哪些方面的优势可以保障项目的成功交付？总之，有管理就必然有度量，而且度量结果应该是数字化的。

Agile中估算&度量的痛点

Agile模式中有传统不曾体会的痛点，而有些痛点则是共同的。让我们看看传统的作为传统研发模式的代表CMMI模型中，可以看到传统项目里有很多的估算和度量指标等着我们去填，还有质量人员（QA）催着。而在敏捷管理类型的项目上（特别是互联网产品），人们更多关注的是软件价值、创新型、业务模式，而弱化了研发的进度成本质量的管理。随着产品进入平稳运营期，我们的视线重新被拉回到了进度成本质量的管理。或许有人会觉得不应该走这回头路，有这样想法的人通常是把产品经理和研发管理混到了一起。作为产品经理，应该始终保持产品的价值设计，而研发管理则应该通过提升研发效率，质量来保障价值的交付。因此对于研发管理，度量从项目开始之处就应该存在，而非等到不能等的那一天。据我了解，京东在产品持续创新的同时，也开始建立每个产品的研发成本和效率进行度量和分析。（换句话说，投资者们需要知道这些产品值不值投入）

Agile估算的计划洋葱



计划类型	估算对象	计量单位
Strategy	价值	市场占有率等
Portfolio		
Product	价值/管理指标	市场占有率，及Story Point
Release	管理指标	Story Point
Iteration		
Day		Hours

千年大计，看开局。Agile计划中的Strategy和Portfolio都是产品价值的估算的结果，如果这里有问题，后面就都白费了。不过在此不做产品价值的深究。而从Product往下都是基于研发管理的估算，在Agile中我们用户故事点为单位，直到Day这层的时候，才用Hour为单位对任务进行估算。这让我们发现，估算点（story point）作为一个非标准计量单位在这里使用的局限性了。我们先看看故事点的优缺点：

故事点估算的优点

我曾经写过一篇《用户故事为何用斐波那契数列》的博文，里面主要解释了斐波那契数列的使用意义。作为用户故事估算方法，这个方法的优点是：渐进明细的原则，通过需求条目化方法，将需求逐步拆分至满足进入迭代目标的要求。其规定的估算数字更是让团队通过故事地图方式迅速达成一致意见，建立初步的规模估算。总结的说：精益的估算方式，快速的估算。

故事点估算的缺点

故事点的核心问题就在于缺乏统一基准。其根本原因是用户故事点法用的是类比估算，而非绝对值估算。这就好比当我们去估算足球大小时，我们用乒乓球或篮球去对比足球大小的倍数关系获得结论，由于参照物的不同，得出的结论肯定存在差异。这在一个团

队，或者产品内还好，但是放到跨产品，跨部门或者跨组织的时候，由于基准的不同，让估算和度量基线设置失去了可比性。除非你始终把估算范围限定在一个产品内，但这显然不符合多产品组合管理的管理目标。所以我经常看到的很多团队逐渐放弃了故事点估算，而最终只采用任务小时估算的原因。

A团队的100Pts > B团队的50Pts ？

但是如果我们不采用故事点估算，新的问题又来了。对需要我们如果直接用工作量进行表示，这个前提就是你清楚理解所需要实现的需求，以及能够拆分出要执行的任务，让有经验的人进行估算才能得出想要的值，而这样做就似曾相识的变回了瀑布模型了，这显然是行不通的。所以要想建立统一的管理度量目标就无从谈起了。

什么是Nesma FPA估算

既然故事点难以当此大任，摆在面前的问题究竟如何解呢？幸好本人代表中国软件协会过改分会，是国际ISBSG（国际度量标准组）组织的理事。发现功能点和敏捷估算是完全可以结合在一起成功解决这一问题。事实上FPA并非新鲜事物，早在1975年IBM的工程师Allan Albrecht提出了FPA方法。Allan的最初目的是为了估算工作量，他发现通过统计打破系统的边界的行为次数就可以建立与工作量之间的联系。他就为系统的这种功能设计出一套估算参数。

后来这个参数基础上又延伸出很多估算方法，目前被国际ISO组织接受的国际标准方法一共有4种：

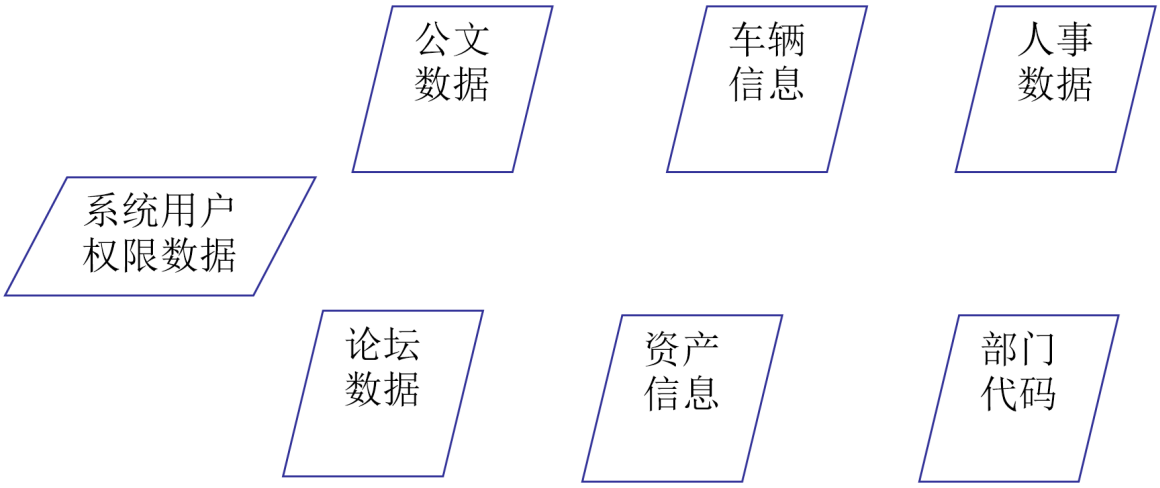
- 国际功能点用户协会提出的IFPUG功能点分析方法。
- 荷兰软件度量协会（NEtherlands Software Metrics Association，NESMA）提出的荷兰软件功能点分析方法。
- 英国软件度量协会（UK Software Metrics Association，UKSMA）提出的Mk II功能分析方法（Mark II FPA）。
- 通用软件度量国际协会（COmmon Software Measurement International Consortium，COSMIC)提出的全功能点分析方法（COSMIC-FFP）。

很久之前业界使用FPA的企业，他们使用的是IFPUG居多，这是第一代估算方法，但是因为其计数复杂，学习成本高，所以并未被国内企业很好的接受和使用。而Nesma的FPA是在此基础上的改进，应对软件不同时期的需求特点，增加了快速估算和详细估算方法，估算参数也做了默认值推荐，让估算人员可以更快掌握这一方法。

Nesma FPA模型

在FPA中功能（需求）的定义和用户故事一样，是从用户角度出发系统进行信息交互的过程。功能分为两类：一个是事务型；第二个是数据型，其区别是维护数据文件和使用数

据文件的职能不同。数据文件可以简单理解为满足第三范式的数据库模型，有多少个数据文件，就是有多少数据型功能。FPA把由外部系统维护数据文件称为ELF（External Logic File），系统内部的数据文件称为ILF（internal Logic File)。数据文件举例：



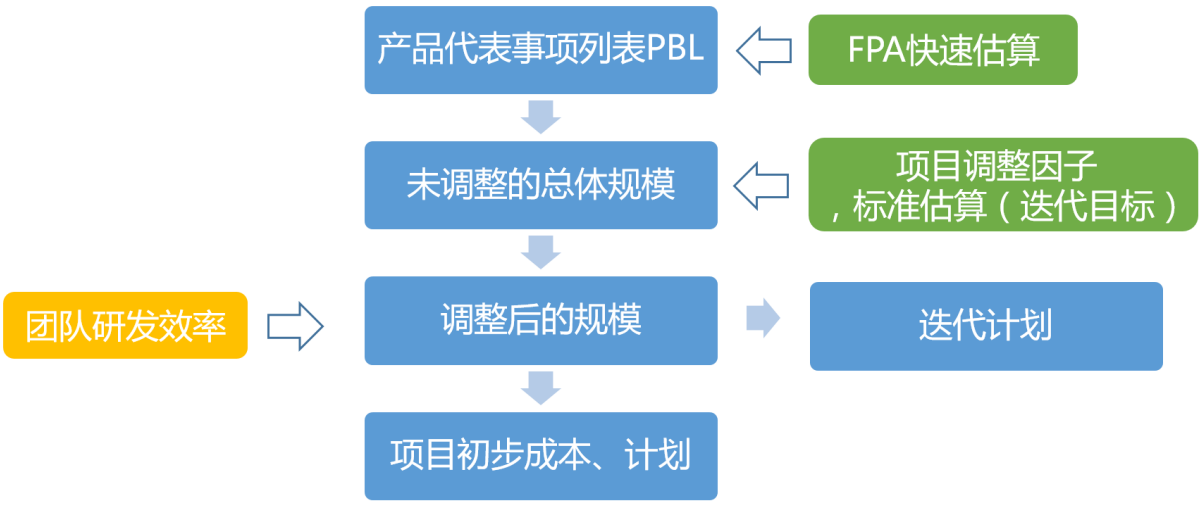
对于系统的输出EO（External Output，如报表），输入EI（External Input，如用户信息登记），查询（external Query，如显示企业所有女员工信息) 则是事务型功能。

软件的估算方式应该是收敛形态，即指望在项目初期就能一次性估算出和实际结果非常精确的值，也不能始终依据一个估值进行计划。正确的做法是在不同的阶段获得估算的范围，随着交付的临近逐步的获得更为精确估算值。我们可以根据历史发生的估算数据和实际数据进行对比，获得一个估算区间。从而为每个项目设定合理的计划安排。随着数据积累的不断完整，我们估算的偏差区间将逐步变小，而这才是我们改进的目标。

收敛的估算

FPA估算方法

让我们先看下FPA结合敏捷，在敏捷计划洋葱中发挥作用，具体过程如下：



产品初步估算

应对不同的需求阶段，NESMA采用不同的估算参数。比如产品初期阶段，需求尚未明确以及拆分，FPA中只计数ILF*35加上ELF*15数据文件文件数即可初步获得软件规模。计算规则如下：

总体UFP（Unadjusted Function Point）=ILF*35+ELF*15

这很好的解决了项目规模的预估工作，即使需求不很明确，但是数据文件在可见范围是可估算的。对比敏捷用户故事点估算的40，100和无穷大等值会精确很多。

这个过程可以应对到Agile的Product级的估算中，根据计算得出的工作量，成本和工期，快速制定出Release Plan。

迭代估算

当需求进入迭代开发周期后，用户故事被进一步拆分明确，对于故事所包含的内容是明确的。这个时候我们需要用到下诉参数表：低，一般，高是应对的每个功能的复杂度（对于功能复杂度在此不做深究，因为跟具体涉及的文件数量和携带字段信息有关，默认情况下，采用默认红色的值即可。只有发现特别复杂/简单的情况下，可以通过另外一个复杂度参数表获得其复杂度，如ELF字段超过50为一般。）

	低	一般	高
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

而Iteration Plan里的迭代规模UFP的计算公式就变为：

总体UFP=7*ILF+5*ELF+4*EI+5*EO+4*EQ

到这里，基本解决了敏捷估算缺乏统一基准的进行估算的问题。而这一步才是度量的开始，因为没有统一基准就无从谈起。

为了更好的理解，这里举个例子计算考核-配置考核计数参数的功能让大家有个初步了解。

用户故事描述

故事描述	作为主管考核人员，我可以配置个险代理人出勤率的管理机构，起始日期，主管考核标准参数， 以便计算时得出相应的出勤率。
界面原型：	

计数结果

事务功能：增加（EI）：修改（EI），查询（EQ）。

数据功能：考核配置数据文件（ILF)，主管考核数据文件（ILF）管理机构数据文件ILF，如果之前功能已经创建，在此将不被重复计数。

起始日期为固定参数，不作为维护数据文件。这样我们得出这个功能的：UFP = 4+4+5+7+7+7 = 34 点。

如果对比快速估算，上面的需求的一句话描述可能是这样的：进行考核参数调整，作为出勤率计算依据。这里就是考核配置数据文件（ILF），UFP=1ILF*35= 35。可以发现估算的偏差是很小的。

从上面这个例子上，我们可以看到其出发点和Agile中对于用户故事的出发点是一致的，即用户视角。另一个共同点是，FPA的功能和用户故事都是独立完整的。例如：考核参数配置基本信息录入功能，对用户来说，一次完整的信息录入保存才算是功能，任何打断保存或者中间状态都不能称为功能（比如分页录入），这跟敏捷中用户故事必须是完整独立的功能描述方式是一致的。基于此FPA就可以无缝的结合用户故事估算所使用。

功能点调整因子（Factors）

我们之前的估算都被称为UFP，这个对于不同的系统，不同的环境因素，要想得出最终工作量还需要几个步骤。事实上，任何因素都会影响研发效率，而我们关注的是最可能影响的几个因素。在FPA中梳理出最容易影响开发的14个因子。如图中设计的：

通用系统特征		得分	通用系统特征		得分
G1	数据通讯	4	G8	在线更新	3
G2	分布式数据处理	3	G9	负责的处理	2
G3	性能	4	G10	可复用性	2
G4	高强度配置	3	G11	易安装性	4
G5	交易速度	2	G12	易操作性	3
G6	在线数据输入	4	G13	多场地	3
G7	最终用户效率	3	G14	支持变更	4

功能点的调整系数是通过通用系统特性及其影响程度来评定的，对每个常规系统特性的评估由其影响程度（DI）而定，分为0 - 5级：

0 毫无影响 1 偶然影响 2 适度影响 3 一般影响 4 重要影响 5 强烈影响 。

然后依次对以下14个系统常规特性进行打分，并带入以下计算公式算出功能点的调整因子。 Value Adjustment Factor=(sum of (DI) * 0.01) + 0.65 ， 默认情况为一般影响，如果因子都偏低，说明项目复杂较低，反之项目比较复杂，整体规模会增加。调整后的规模FP = UFP * VAF .
如果将这次调整因子对应到上面的例子中：
FP=34*（44*0.01+0.65）= 37.1

基于FPA的敏捷度量基本维度

敏捷的度量同样围绕：成本，进度，质量。基于FPA，我们可以做到统一化的度量方式。除了敏捷自带的跟踪方式外。我们可以增加组织级的度量维度如下：

组织级估算：

指标名称	度量目的	指标定义/算法	分析方法
预算成本	预估项目的研发成本，控制成本风险	=FP*人均成本*人均功能点	历史数据统计人均成本和人均功能点
组织生产率	对组织级的生产效率进行评价	=完成功能点/季度	与基线比对，分析异常点
人均产能	反映组织、团队和室的产出能力	=（已完成项目折算功能点+实施中项目已投入估算功能点）/统计人月	观察趋势变化，同比、环比比较
测试速率	反映ST、UAT测试执行效率	=项目测试用例/功能点	观察趋势变化，同比、环比比较
质量	反映ST、UAT测试执行的有效性	=缺陷数/功能点	观察趋势变化，同比、环比比较
供应商效能	反应外包供应商的交付能力	人均产能/人均单价	同行对比

敏捷产品级度量（Scrum交付模式）

效率：

敏捷效率度量

敏捷度量项	描述
迭代燃尽图	每个迭代的研发进度管理
团队生产率（Velocity）	团队每个周期的交付能力
发布燃尽图(Burndown Chart)	与计划剩余工作量比较，判断总体进度是否落后或者超前
发布燃烧图(Burn-Up Chart)	累计完成的工作总量之和
迭代交付周期	每次固定的交付周期

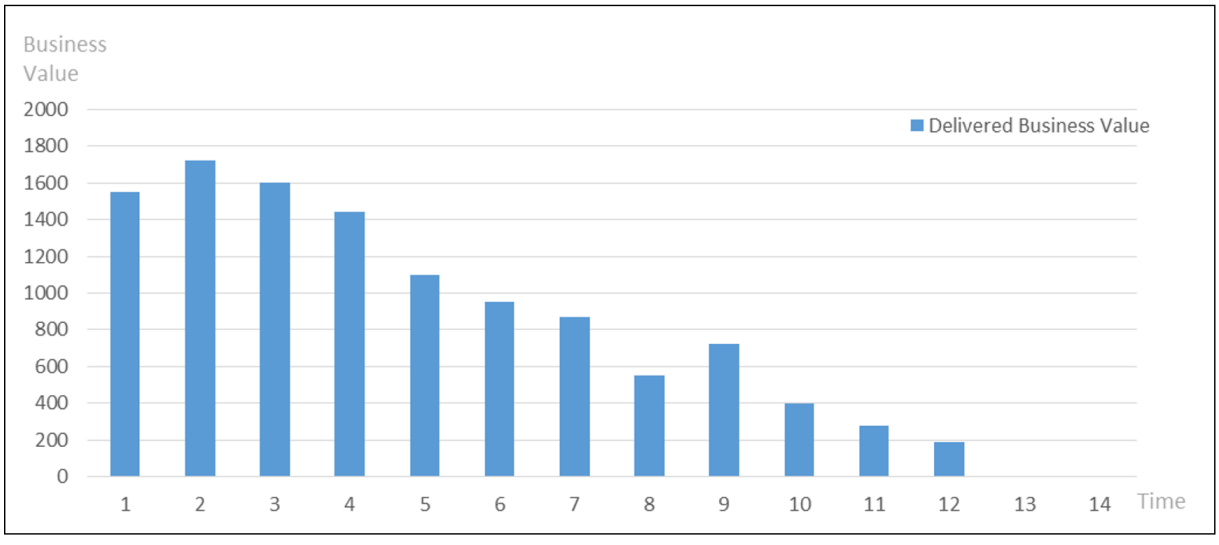
质量：

敏捷质量度量

度量项	描述
缺陷泄露率	从研发泄漏到生产环境的缺陷数量
分层自动化测试覆盖率	为系统建立三层自动化测试所包含的每层的测试覆盖率。
缺陷修复周期	从发现缺陷到修复缺陷的平均周期，表示团队对于问题的响应速度

价值交付：

敏捷-业务价值交付图



团队信心：

团队交付信心指数

	Jul 1	Jul 2	Jul 3	Jul 4	Jul 5	Jul 6
Tom	☺	☺	☺	😐		
Dick	☺	😐	😐	☹		
Harry	☹	☺	☺	☺		
Marry	☹	☺	😐	☺		
Jim	☺	☺	☹	☺		
Christie	😐	☺	☹	☹		

总结

业界研究敏捷估算和度量的人不多，说明更多人还关注在而不是改进， 这其实是悲哀的。一个好的方法的生命力在于其对组织的价值，回到开篇**“人是无法管理自己不了解的事物”**这句话，如果我们对于敏捷方法不能很好的进行度量管理，敏捷的持续改进就无从说起了。敏捷转型并不仅是方法的导入，而是改进文化的建设和实践的深入。脱离实际讲理论那是”虚“，脱离理论搞落地那是“蛮干”。我们正确的做法是在理论指导下，通过实践获得数字化度量的反馈，保持不断改进。期盼各位读者共同保持敏捷改进的热情！