Atitit it业界与软件界的定律 原则 准则 法则 效应

[1.1. 一切都是管理问题定律 ati总结 2](#_Toc28674)

[1.2. 一万小时定律 专家定律 2](#_Toc30630)

[1.3. 定律 布鲁克斯定律：      人月=人\*月，月≠人月/人 2](#_Toc18295)

[1.4. Conway’ s Law”（康威定律 组织架构影响产品与项目定律 2](#_Toc29822)

[1.5. 摩尔定律（Moore's Law）、 3](#_Toc26087)

[1.6. 安迪——比尔定律（Andy and Bill's Law） 4](#_Toc32349)

[1.7. 和反摩尔定律（Reverse Moore's Law）。这 4](#_Toc11353)

[1.8. 软件开发定律系列之1:3:9定律 4](#_Toc17402)

[1.9. 最小功效原则 4](#_Toc16710)

[2. 五大定律：软件开发中的时间估算.html 4](#_Toc16901)

[2.1. 估算第一定律：估算纯属浪费 5](#_Toc18005)

[2.2. 估算第二定律：估算不可互换 5](#_Toc5232)

[2.3. 估算第三定律：估算是错的 5](#_Toc3307)

[2.4. 估算第四定律：估算是暂时的 5](#_Toc24392)

[2.5. 估算第五原则：估算是必要的 5](#_Toc18486)

[3. 软件开发中的11个系统思维定律 5](#_Toc29450)

[3.1. 1. 今日的问题源于昨日的解决方案（Today’s problems come from yesterday’s solutions） 6](#_Toc28835)

[3.2. 2. 用力越大，系统的反作用力也越大（The harder you push, the harder the system pushes back） 6](#_Toc12495)

[3.3. 3. 福兮祸之所伏（Behavior grows better before it grows worse） 6](#_Toc27605)

[3.4. 4. 最容易出去的方法往往会导致返回来（The easy way out usually leads back in） 6](#_Toc5330)

[3.5. 5. 治疗带来的结果可能会比疾病导致后果更严重（The cure can be worse than the disease） 6](#_Toc26016)

[3.6. 6. 欲速则不达（Faster is slower） 6](#_Toc16915)

[3.7. 7. 在时间和空间上，因果并不密切相关（Cause and effect are not closely related in time and space） 6](#_Toc27030)

[3.8. 8. 微小的改变可以产生明显的效果，但这种杠杆效应最大的地方往往也最不明显（Small changes can produce big results-but the areas of highest leverage are often the least obvious） 6](#_Toc37)

[3.9. 9. 鱼与熊掌可以兼得，但不是同时兼得（You can have your cake and eat it too – but not at once） 7](#_Toc2590)

[3.10. 10. 把一头大象分两半不会得到两头大象（Dividing an elephant in half does not produce two small elephants） 7](#_Toc25712)

[3.11. 11. 无可非议（There is no blame） 7](#_Toc2357)

[4. 软件项目开发七大定律 7](#_Toc27237)

[4.1. 1. 1:10:100 定律：需求错误导致的成本是修复程序错误成本的 100 倍。 反 7](#_Toc22599)

[4.2. 1:2 定律：在开发中，每花费 1 美元，在维护中就得花费 2 美元，因此要注意度量改进 维护的度量元。 7](#_Toc6780)

[4.3. 3. Weinberg 可靠性零定律： 7](#_Toc12227)

[4.4. 4. 1:3:9 定律： 8](#_Toc7663)

[4.5. 帕金森定律（Parkinson’sLaw） ：工作总是用完所有可利用的时间。 9](#_Toc17492)

[4.6. 布鲁克斯定律（Brooks’Law） ：人月=人\*月，但是月≠人月/人，投入更多的人到一项 延迟的工作上，可以导致该项工作更加延迟。 9](#_Toc19862)

[4.7. 7. 80-20 定律：80%的错误集中于 20%的模块；80%的错误来自于 20%的人员； 9](#_Toc3819)

[5. 其他定律 9](#_Toc31821)

[5.1. Atwood法则：凡是可以用javascript实现的，最终都会用javascript实现 9](#_Toc1526)

[5.2. 最小功效原则 10](#_Toc450)

[6. 参考 12](#_Toc22906)

## 一切都是管理问题定律 ati总结

项目的延迟就是管理的问题

## 一万小时定律 专家定律

## 定律 布鲁克斯定律：      人月=人\*月，月≠人月/人

## Conway’ s Law”（康威定律 组织架构影响产品与项目定律

1967年，《哈佛商业评论》拒绝了 Mel Conway 提交的一篇论文。一年之后，Conway 的论文最终被确定为 “Conway’ s Law”（康威定律）。康威在加利福尼亚理工学院获得物理学硕士学位，在凯斯西储大学获得数学博士学位。毕业之后，他参与了很多知名的软件项目，如 Pascal 编辑器。在他的职业生涯中，康威观察到一个现象：软件团队开发的产品是对公司组织架构的反映。

## 摩尔定律（Moore's Law）、

科技行业流传着很多关于比尔·盖茨的故事，其中一个是他和通用汽车公司老板之间的对话。盖茨说，如果汽车工业能够像计算机领域一样发展，那么今天，买一辆汽车只需要25美元，一升汽油能跑400公里。通用汽车老板反击盖茨的话我们暂且不论，这个故事至少说明计算机和整个IT行业的发展比传统工业要快得多。

人们多次怀疑摩尔定律还能适用多少年，就连摩尔本人一开始也只认为IT领域可以按这么高的速度发展10年，至于以后当时他也说不清了。而事实上，从二战后至今，IT领域的技术进步一直是每一到两年翻一番，至今看不到停下来的迹象。至少，在我第一次刊登这篇博客的2007年到2011年的今天，摩尔定律依然适用。在人类的文明史上，没有任何一个其他行业做到了这一点。因此，IT行业必然有它的特殊性。

## 安迪——比尔定律（Andy and Bill's Law）

## 和反摩尔定律（Reverse Moore's Law）。这

## 软件开发定律系列之1:3:9定律

## 最小功效原则

互联网之父Tim Berners-Lee 在他的著作 《最小功效原则》中的一段话：

“在过去的40年中，计算机科学一直在开发尽可能功能强大的语言。现在我们必须领会这样的道理：要选择功能最不强的解决方案而不是最强的。语言的功效越小，对于储存在该语言中的数据你能做的事情就越多。如果你把程序写成简单的描述性形式，任何人都可以编写一个程序来分析它。比如，如果一个包含了天气数据的网页，里面使用RDF 格式来描述这些数据，用户就可以把它当做一个表格来查找，也许求它的平均值，或者用它绘图，或把它和其他信息结合在一起进行推理分析。另一个极端情况是，这些天气信息是用漂亮的Java applet描绘出来的。虽然这样可能做出很酷的用户界面，但它完全无法用于分析。找到这个页面的搜索引擎会压根不知道这些数据是什么或者是干什么用的。唯一能发现Java applet的含义的方式就是让它在一个人面前运行起来。”

# 五大定律：软件开发中的时间估算.html

## 估算第一定律：估算纯属浪费

## 估算第二定律：估算不可互换

## 估算第三定律：估算是错的

## 估算第四定律：估算是暂时的

## 估算第五原则：估算是必要的

# 软件开发中的11个系统思维定律

　以上11条系统思维定律表明，我们提出的所有解决方案都会产生一定的后果，有时非常严重并出乎意料。我们周围的系统本就那样，我们不应苛责它们，而是要从中学习。要掌握系统思维方式并控制这些系统，我们需要做到如下几点：  
　　1. 要明白我们是在跟什么样的系统打交道，是人或是软件；  
　　2. 有意识地学习相互关系、因果链；  
　　3. 把系统看做一个整体，并且视其为其他系统的一部分。  
　　系统思维方面有很多挑战，通过获取并且利用有关系统工作方式的知识，我们可以战胜其中的很多挑战。但是，大部分严峻挑战是我们人类与之相冲突的本性。我们的激情、感情以及本能可以轻易改变我们理智、条理分明的思维方式。掌握系统思维方式的第一步就是要学习如何跟自己合作。

## 1. 今日的问题源于昨日的解决方案（Today’s problems come from yesterday’s solutions）

## 2. 用力越大，系统的反作用力也越大（The harder you push, the harder the system pushes back）

## 3. 福兮祸之所伏（Behavior grows better before it grows worse）

## 4. 最容易出去的方法往往会导致返回来（The easy way out usually leads back in）

## 5. 治疗带来的结果可能会比疾病导致后果更严重（The cure can be worse than the disease）

## 6. 欲速则不达（Faster is slower）

## 7. 在时间和空间上，因果并不密切相关（Cause and effect are not closely related in time and space）

## 8. 微小的改变可以产生明显的效果，但这种杠杆效应最大的地方往往也最不明显（Small changes can produce big results-but the areas of highest leverage are often the least obvious）

## 9. 鱼与熊掌可以兼得，但不是同时兼得（You can have your cake and eat it too – but not at once）

## 10. 把一头大象分两半不会得到两头大象（Dividing an elephant in half does not produce two small elephants）

管理层承诺，每发现一处系统bug，测试者将得到5美元。测试者对跟开发者合作不再感兴趣，并且不再试图消除产生bug的根本因素。团队之间良好而且高效的关系不复存在。

## 11. 无可非议（There is no blame）

# 软件项目开发七大定律

## 1. 1:10:100 定律：需求错误导致的成本是修复程序错误成本的 100 倍。 反

## 1:2 定律：在开发中，每花费 1 美元，在维护中就得花费 2 美元，因此要注意度量改进 维护的度量元。

反思： ? ? 在我们公司的项目中维护成本与开发成本的比例是多少？ 我们在需求开发、设计过程中为了降低维护的成本采取了哪些措施？

## 3. Weinberg 可靠性零定律：

如果一个系统不要求是可靠的，那么它能够满足任何的其他 目的；换句话说，如果对实际工作的程序没有要求，那么你能满足任何设置的编码交付 期。 反思： ? 在限定了资源， 而项目工期又比较紧张时， 我们通常牺牲了什么？我们是否真的加 快了进度呢？ ? 成功的项目管理追求四要素的平衡，即需求、质量、进度、资源。

## 4. 1:3:9 定律：

随着软件系统规模的增大，其成本成倍增长，呈现 1:3:9 的关系，称之为 软件产业的非规模经济现象，举个例子：制药行业，只要做好配方，一条流水线就可以 生产几吨的成品药，药的成本微乎其微；相较定制化的软件来说，面对的用户越多，系 统功能越多，产生的需求和缺陷就越多，开发成本也成倍数增长。 反思： ? ? ? 5. 我们如何降低软件的开发成本？ 为什么提倡采用迭代的生命周期模型？ 为什么提倡小项目、小团队？

## 帕金森定律（Parkinson’sLaw） ：工作总是用完所有可利用的时间。

如果你给自己安 排了充裕的时间从事一项工作， 你会放慢你的节奏以便用掉所有分配的时间； 容易达到 的目标将使员工工作上变得松懈。 反思： ? ? 如何规避帕金森定律？ 如果整个项目有 20%的缓冲时间，你会如何分配这 20%的缓冲？

## 布鲁克斯定律（Brooks’Law） ：人月=人\*月，但是月≠人月/人，投入更多的人到一项 延迟的工作上，可以导致该项工作更加延迟。

1. Barry Boehm 认为：可以将软件开发进度压缩 25%，但是不能再多了； 200/20/6X 现象：人数增加 1 倍，工期缩短 20%，缺陷增加 6 倍。 反思： ? ? 在实践中，我们是否经常通过给项目组增加人手的方式加快进度？ 有哪些合理的加快进度的措施？

## 7. 80-20 定律：80%的错误集中于 20%的模块；80%的错误来自于 20%的人员；

1. 80%的 错误集中于 20%的类型；80%的软件废品和返工是由 20%的缺陷引起的；80%的资源 是由 20%的构件消耗的；80%的工程活动是通过 20%的工具完成的；80%的进展是 20%的人完成的。 反思：

# 其他定律

## Atwood法则：**凡是可以用javascript实现的，最终都会用javascript实现**

**十年来，编程领域有什么重要进展？ - caoglish 的回答 - 知乎.html**

## 最小功效原则

互联网之父Tim Berners-Lee 在他的著作 《最小功效原则》中的一段话：

“在过去的40年中，计算机科学一直在开发尽可能功能强大的语言。现在我们必须领会这样的道理：要选择功能最不强的解决方案而不是最强的。语言的功效越小，对于储存在该语言中的数据你能做的事情就越多。如果你把程序写成简单的描述性形式，任何人都可以编写一个程序来分析它。比如，如果一个包含了天气数据的网页，里面使用RDF 格式来描述这些数据，用户就可以把它当做一个表格来查找，也许求它的平均值，或者用它绘图，或把它和其他信息结合在一起进行推理分析。另一个极端情况是，这些天气信息是用漂亮的Java applet描绘出来的。虽然这样可能做出很酷的用户界面，但它完全无法用于分析。找到这个页面的搜索引擎会压根不知道这些数据是什么或者是干什么用的。唯一能发现Java applet的含义的方式就是让它在一个人面前运行起来。”

## CAP数据技术的理论基础

（Consistency一致性、Availability可用性、Partition-tolerance分区可容忍性）理论普遍被当作是大数据技术的理论基础。同时，根据该理论，业界有一种非常流行、非常“专业”的认识，那就是：关系型数据库设计选择了C（一致性）与A（可用性），NoSQL数据库设计则不同。其中，HBase选择了C（一致性）与P（分区可容忍性），Cassandra选择了A（可用性）与P（分区可容忍性）。

? 在实践中，我们应该如何运用 80-20 定律？ 每一条定律的反思都值得我们在实际项目管理过程中关注。

# 参考

## Atitit 生活总常见定律 原则 准则

## Atitit 管理学心里学的50大定律

## 计算机行业三大发展定律\_百度文库.html

## 如何正确理解CAP理论？ - 大树叶 技术专栏 - CSDN博客.mhtml

## 计算机行业的三大发展定律-CSDN.NET.html

## 五大定律：软件开发中的时间估算.html

## 软件开发中的11个系统思维定律\_知识库\_博客园.html

## 浅谈软件开发定律系列之1\_3\_9定律 - 柳记 - 51CTO技术博客.html

## 软件长寿法则，记住这7条 - 研发管理 - ITeye资讯.html ( ned detail)

## 软件项目七定律引起的反思\_百度文库.html

## 康威定律\_创业公司的组织架构是如何影响产品的\_-频道-手机搜狐.mhtml

作者:: 绰号:老哇的爪子claw of Eagle 偶像破坏者Iconoclast image-smasher

捕鸟王"Bird Catcher kok 虔诚者Pious 宗教信仰捍卫者 Defender Of the Faith. 卡拉卡拉红斗篷 Caracalla red cloak 万兽之王 纵火者

简称：： Emir Attilax Akbar 埃米尔 阿提拉克斯 阿克巴

全名：：Emir Attilax Akbar bin Mahmud bin attila bin Solomon bin adam Al Rapanui 埃米尔 阿提拉克斯 阿克巴 本 马哈茂德 本 阿提拉 本 所罗门 本亚当 阿尔 拉帕努伊

常用名：艾提拉（艾龙）， EMAIL:1466519819@qq.com

头衔：uke总部o2o负责人，全球网格化项目创始人，

uke交友协会会长 uke捕猎协会会长 Emir Uke部落首席大酋长，

uke宗教与文化融合事务部部长， uke制度与重大会议委员会委员长，uke保安部首席大队长,uke制度检查委员会副会长，

UTSC uke技术标准化委员会委员长 uke 首席cto 软件部门总监 技术部副总监 研发部门总监主管 产品部副经理 项目部副经理 uke科技研究院院长 uke软件培训大师

uke波利尼西亚区大区连锁负责人 汤加王国区域负责人 uke克尔格伦群岛区连锁负责人，莱恩群岛区连锁负责人，uke布维岛和南乔治亚和南桑威奇群岛大区连锁负责人

Uke软件标准化协会理事长理事长 Uke 数据库与存储标准化协会副会长

uke终身教育学校副校长 Uke医院 与医学院方面的创始人

uec学院校长， uecip图像处理机器视觉专业系主任 uke文档检索专业系主任

Uke图像处理与机器视觉学院首席院长

Uke 户外运动协会理事长 度假村首席大村长 uke出版社编辑总编

转载请注明来源：attilax的专栏 <http://blog.csdn.net/attilax>

<http://www.cnblogs.com/attilax/>

Microblog

<http://weibo.com/u/5941179815> (common)

<http://weibo.com/u/5487832265>

<http://weibo.com/u/5487832265> (tech)

Qq 1466519819 微信attilax

--Atiend v12