Atitit 软件方法论法典概论 attilax著作

目录

[1. 基本法通则总则 2](#_Toc17499)

[2. 流程方法 3](#_Toc18051)

[2.1. 工具链建设法 4](#_Toc23358)

[2.2. 代码编写法 4](#_Toc2982)

[2.3. 注释法 4](#_Toc11796)

[2.4. 文档法 4](#_Toc28043)

[2.5. 源码管理法 4](#_Toc13777)

[2.6. 开发法 测试法 设计法 需求法 4](#_Toc6193)

[2.7. Review法 4](#_Toc4588)

[2.8. 用户手册、安装手册、项目总结报告 4](#_Toc2779)

[2.9. 技术选型法 4](#_Toc21403)

[2.10. Debug trace 法 4](#_Toc27872)

[3. 常见关系法 4](#_Toc23534)

[3.1. 数据使用法 4](#_Toc24032)

[3.2. 框架法 类库法 5](#_Toc11175)

[3.3. 接口法 5](#_Toc17579)

[3.4. 代理法 5](#_Toc15114)

[3.5. 组件法 5](#_Toc31327)

[3.6. 封装法 5](#_Toc24825)

[3.7. 集成法 5](#_Toc23437)

[3.8. 继承法 Atitit uke it uke继承法 5](#_Toc19477)

[3.9. Invoke法 5](#_Toc22371)

[3.10. 接口实现法 5](#_Toc26042)

[4. 重大问题解决法 5](#_Toc5411)

[4.1. 功能不全 5](#_Toc30225)

[4.2. 性能 稳定性 5](#_Toc19066)

[4.3. Ux体验 6](#_Toc22933)

[4.4. 开发效率 6](#_Toc29045)

[4.5. 可我维护 可扩展 6](#_Toc23471)

[4.6. 运行错误解决法 6](#_Toc17184)

[5. 争议解决法 6](#_Toc10080)

[6. 功能实现法 6](#_Toc17428)

[6.1. 图像处理法 6](#_Toc1554)

[6.2. 通讯选型法 6](#_Toc23164)

[6.3. 存储选型法 6](#_Toc9727)

[6.4. Ui选型法 6](#_Toc24933)

[6.4.1. 准则 6](#_Toc11478)

[7. 其他 7](#_Toc2890)

# 基本法通则总则

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

1.2. 一万小时定律 专家定律 2

1.3. 定律 布鲁克斯定律：      人月=人\*月，月≠人月/人 2

1.4. Conway’ s Law”（康威定律 2

1.5. 摩尔定律（Moore's Law）、 2

1.6. 安迪——比尔定律（Andy and Bill's Law） 3

1.7. 和反摩尔定律（Reverse Moore's Law）。这 3

1.8. 软件开发定律系列之1:3:9定律 3

1.9. 最小功效原则 3

2. 五大定律：软件开发中的时间估算.html 4

2.1. 估算第一定律：估算纯属浪费 4

2.2. 估算第二定律：估算不可互换 4

2.3. 估算第三定律：估算是错的 4

2.4. 估算第四定律：估算是暂时的 4

2.5. 估算第五原则：估算是必要的 4

3. 软件开发中的11个系统思维定律 4

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

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

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

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

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

3.6. 6. 欲速则不达（Faster is slower） 6

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

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

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

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

3.11. 11. 无可非议（There is no blame） 7

4. 软件项目开发七大定律 7

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

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

4.3. 3. Weinberg 可靠性零定律： 7

4.4. 4. 1:3:9 定律： 8

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

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

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

5. 参考 11

Atitit v2快速开发的 大原则.docx

Atitit 提升进度的大原则与方法 高层方法 attilax总结.docx

# 流程方法

atitit 项目环境搭建法.docx

Atitit 架构的原则attilax总结 v3 r10 - 副本.docx.txt

## 工具链建设法

## 代码编写法

## 注释法

## 文档法

## 源码管理法

## 开发法 测试法 设计法 需求法

## Review法

## 用户手册、安装手册、项目总结报告

## 技术选型法

## Debug trace 法

## 模块管理法

# 常见关系法

主体之间的关系的规范

## 数据使用法

## 框架法 类库法

## 接口法

## 代理法

## 组件法

## 封装法

## 集成法

## 继承法 Atitit uke it uke继承法

## Invoke法

## 接口实现法

# 重大问题解决法

## 功能不全

## 性能 稳定性

Atitit 性能指标与性能提升的5个原则与性能提升模型.docx

## Ux体验

## 开发效率

## 可我维护 可扩展

## 运行错误解决法

# 争议解决法

# 功能实现法

## 图像处理法

## 通讯选型法

## 存储选型法

## Ui选型法

### 准则

减少用户的认知负担

保持[界面](https://baike.baidu.com/item/%E7%95%8C%E9%9D%A2" \t "https://baike.baidu.com/item/GUI/_blank)的一致性

满足不同目标用户的创意需求

用户[界面](https://baike.baidu.com/item/%E7%95%8C%E9%9D%A2" \t "https://baike.baidu.com/item/GUI/_blank)友好性

图标识别平衡性

图标功能的一致性

建立[界面](https://baike.baidu.com/item/%E7%95%8C%E9%9D%A2" \t "https://baike.baidu.com/item/GUI/_blank)与用户的互动交流

更为人性化的视觉优化

更具识别性的图标及其他元素

更具可操控性和扩充性的使用易用性

更具有企业品牌特色的视觉识别性

## Ui渲染法 Atitit 模板引擎, 数据绑定 ui渲染Handlebars

# 其他