**javascript编码规范**

**编号：XZSKC-21**

**编写： 罗娇 日期： 2022/2/4**

**审核： 朱绍攀 日期： 2022/2/13**

**批准： 朱绍攀 日期： 2022/2/13**

**徐州市勘察测绘研究院有限公司**

**文档修订记录**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 版本号 | \*变化  状态 | 简要说明 | 日期 | 变更人 | 批准日期 | 批准人 |
| V1.0 | C | 创建 | 2022/2/2 | 罗娇 | 2022/3/19 | 朱绍攀 |
| V1.1 | M | 完善代码规范 | 2023/1/11 | 罗娇 | 2023/1/11 | 朱绍攀 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

\*变化状态：C = 创立，A = 增加，M = 修改，D = 删除

**目 录**

[1. 概述 1](#_Toc59177087)

[1.1 文档目的 1](#_Toc59177088)

[1.2 适用范围 1](#_Toc59177089)

[2. 代码规范 1](#_Toc59177090)

[2.1 源代码规范 1](#_Toc59177092)

[2.1.1 通用规范 1](#_Toc59177096)

[2.1.2 JavaScript 1](#_Toc59177097)

[2.1.3 文本格式 3](#_Toc59177098)

[2.2 开发原则 3](#_Toc59177099)

[2.2.1 类、接口 3](#_Toc59177100)

[2.2.2 方法 4](#_Toc59177101)

[2.2.3 表达式与语句 5](#_Toc59177102)

[2.2.4 循环语句 6](#_Toc59177103)

[2.2.5 异常捕捉 7](#_Toc59177104)

[2.2.6 变量 8](#_Toc59177105)

[2.2.7 方法 9](#_Toc59177106)

[2.2.8 日志 10](#_Toc59177107)

[2.2.9 其它 10](#_Toc59177108)

[2.2.10 编程技巧 10](#_Toc59177109)

[2.3 代码示例 12](#_Toc59177110)

[2.3.1 Javascript 12](#_Toc59177116)

[3. 研发过程 14](#_Toc59177117)

[3.1 研发计划 14](#_Toc59177121)

[3.1.1 工作描述 14](#_Toc59177122)

[3.1.2 交付工作产品 15](#_Toc59177123)

[3.1.3 文档编写指南 16](#_Toc59177124)

[3.1.4 经验分享 17](#_Toc59177125)

[3.1.5 方法与工具 17](#_Toc59177126)

[3.1.6 自检要素 19](#_Toc59177127)

[3.2 设计与开发 20](#_Toc59177128)

[3.2.1 工作描述 20](#_Toc59177129)

[3.2.2 交付工作产品 20](#_Toc59177130)

[3.2.3 文档编写指南 20](#_Toc59177131)

[3.2.4 经验分享 21](#_Toc59177132)

[3.2.5 方法与工具 21](#_Toc59177133)

[3.2.6 自检要素 23](#_Toc59177134)

[3.3 发版 24](#_Toc59177135)

[3.3.1 工作描述 24](#_Toc59177136)

[3.3.2 交付工作产品 24](#_Toc59177137)

[3.3.3 文档编写指南 24](#_Toc59177138)

[3.3.4 经验分享 24](#_Toc59177139)

[3.3.5 自检要素 25](#_Toc59177140)

[3.4 项目结项 25](#_Toc59177141)

[3.4.1 工作描述 25](#_Toc59177142)

[3.4.2 交付工作产品 26](#_Toc59177143)

[3.4.3 文档编写指南 26](#_Toc59177144)

[3.4.4 经验分享 26](#_Toc59177145)

[3.4.5 自检要素 27](#_Toc59177146)

[4. 例行工作 28](#_Toc59177147)

[4.1 例会及报告 28](#_Toc59177149)

[4.1.1 项目例会 28](#_Toc59177150)

[4.1.2 个人周报 28](#_Toc59177151)

[4.1.3 项目周报 28](#_Toc59177152)

[4.2 研发管理 28](#_Toc59177153)

[4.2.1 里程碑报告 28](#_Toc59177154)

[4.2.2 项目评审 29](#_Toc59177155)

[4.2.3 风险管理 29](#_Toc59177156)

[4.3 项目变更 30](#_Toc59177157)

[4.3.1 变更控制的要点 30](#_Toc59177158)

[4.3.2 如何进行变更 30](#_Toc59177159)

[5. 资料管理 31](#_Toc59177160)

[5.1 文档管理 31](#_Toc59177166)

[5.1.1 日常资料备份 31](#_Toc59177173)

[5.1.2 项目资料归档 32](#_Toc59177174)

[5.2 代码管理 32](#_Toc59177175)

[5.2.1 岗位划分 32](#_Toc59177176)

[5.2.2 版本划分 33](#_Toc59177177)

[5.2.3 流程说明 33](#_Toc59177178)

[5.3 安全管理 34](#_Toc59177179)

# 概述

## 文档目的

为加强对研发总部及各研发分中心的管理、提高研发工作效率、规范开发工作，特制定研发标准及规范。

文档中对研发标准及规范进行详细描述，在研发经验总结、研发项目总结基础上，不断完善规范的有效性，提高项目文档质量，帮助推进项目，保证项目质量，提升研发项目生产效率，确保项目研发成功。

## 适用范围

适用于全体研发工作人员；适用于公司所有研发项目。

# 代码规范



## 源代码规范



### 通用规范

* 开发人员的开发环境应保持统一：（Nodejs，npm，VisualStudio Code，Chrome）。
* 版本控制环境，统一使用公司的Gitlab，研发分中心通过VPN接入。
* 页面设计过程中，要考虑到操作系统、浏览器、浏览器版本、显示器分倍率等因素。
* 代码整洁、注释合理,严格遵守Javascript文档注释标准。

### JavaScript

1. **类/接口**

每个Javascript类，保存为单独的文件（内部类除外），不允许使用非公有类，确有需要的，使用其他修饰方法解决。类名都采用大驼峰风格来命名。

文件开头加入版权声明

|  |
| --- |
| /\*   \* Licensed Materials - Property of tenxcloud.com   \* (C) Copyright 2016 TenxCloud. All Rights Reserved.   \* v0.1 - 2020-10-25   \* @author tianzhiqiang   \*/ |

1. **包(Package)**

在公司整体框架的基础上，每个功能模块单独建立一个包，再按小模块划分，自行建立子包。

包名统一小写，具有业务意义。

1. **方法**

方法的命名,要求遵守：首字母小写，以后每单词首字母大写。功能性方法，以其功能的英文开头，比如：获取jira的项目类别：getJiraProjectCategory。

如果方法名没有合适的英文描述，可使用拼音全拼。如果拼音全拼太长，可适当缩短为拼音首字母，但是要有详细注释。

方法注释：以简明扼要的语句，描述方法所做的事，调用需要注意的事项。各参数含义，返回值。

方法内各语句的注释，以适量为准，能让不了解业务的人看明白。

1. **字段**

字段的命名：以业务规则命名。好的命名，可以代替注释。

字段的注释：以“//”注释在字段声明同一行结尾。

1. **单元测试**

针对每一个业务方法，要求提供mocha的单体测试代码

### 文本格式

* 请以2个空格设置Tab键宽度，按代码层级设置缩进。
* 花括号的使用，左大括号都放在语句开始的这一行。
* 在适当的位置加入空格和空行，以保持代码的美观和可读性。
* 保持适当的折行，一行代码的长度不要超过80个字符。
* 文件的编码格式，统一为UTF-8（包括数据库编码）。

## 开发原则

### 类、接口

* 类的划分粒度，不可太大，造成过于庞大的单个类，也不可太细，从而使类的继承太深。一般而言，一个类只做一件事；另一个原则是根据每个类的职责进行划分，比如用JiraPreference来存放jira偏好信息，而用JiraPreferenceDao来对jira偏好信息进行数据访问操作（比如存取数据库），用JiraPreferenceService来封装jira偏好信息的业务操作等等。
* 多使用设计模式，随时重构。 多个类中使用相同方法时将其方法提到一个接口中，尽量提高重用度。同时可以减少维护难度。
* 每个项目文件中必须要有一个类似indes.js的主类，它提供测试和演示功能
* 在一个单独的应用程序中，主类应该与其他类分开。原因：如果主类和其他类不分开，将会导致很难再使用
* 为你经常使用的类做一个模板。原因：代码标准的一致化。
* 建议多使用接口。原因：接口更容易维护与扩展。
* 为每一个类定义一个默认的构造器，以便于可以构造实例。子类的构造函数内一定要引用父类的一个构造函数。
* 类名都采用大驼峰风格来命名。

### 方法

* 一个方法只完成一项功能，在定义系统的公用接口方法外的方法应尽可能的缩小其可见性。
* 避免用一个类是实例去访问其静态变量和方法。
* 避免在一个较长的方法里提供多个出口，如：

|  |
| --- |
| //不要使用这钟方式，当处理程序段很长时将很难找到出口点  if(condition){  return A;  }  else{  return B;  }  //建议使用如下方式  String result = null;  if(condition){  result = A;  }  else{  result = B;  }  return result; |

* 避免过多的参数列表

尽量控制在5个以内，若需要传递多个参数时，应使用一个容纳这些参数的对象进行传递，以提高程序的可读性和可扩展性。 参数类型和返回值尽量接口化，以屏蔽具体的实现细节，提高系统的可扩展性，

例如：

getProjectAnalysis(projectKey) {}

getProjectList(params) {}

### 表达式与语句

* 控制语句

1.判断中如有常量，则应将常量置与判断式的右侧。

如：

if (isAdmin() == true)...

if (user == null)...

if (strParameter == ‘’) ... 这种判断还可以省了判断 null

2.尽量不使用三目条件判断。

3.判断条件中，尽量使用严格运算符，即用 === 而不是 ==，原因是严格运算符能判断两边值与类型都相同，从而是判断逻辑更加严密。

4.所有if语句必须用{}包括起来,即便是只有一句，如：

|  |
| --- |
| //不要使用这种  if (true) i = 0;  //使用这种方式  if (true){  i = 0;.  } |

* 过多的else分句请将其转成switch语句或使用子函数

每当一个case顺着往下执行时(因为没有break语句)，通常应在break语句的位置添加注释。如：

|  |
| --- |
| switch (condition) {  case ‘a’:  //statements;  //继续下一个CASE  case ‘b’:  //statements;  break;  case ‘c’:  //statements;  break;  default:  //statements;  break;  }//end switch |

### 循环语句

循环中必须有终止循环的条件或语句，避免死循环。

当在for语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在for循环之前(为初始化子句)或for循环末尾(为更新子句)使用单独的语句。

避免在循环中执行取固定值的重复操作，比较一下两种循环的差异：

|  |
| --- |
| //不推荐方式\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  while(index < products.getCount()){  //每此都会执行一次getCount()方法，  //若此方法耗时则会影响执行效率  //而且可能带来同步问题，若有同步需求，请使用同步块或同步方法  }  //推荐方式\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  //将操作结构保存在临时变量里，减少方法调用次数  let count = products.getCount();  while(index < count){  } |

建议将循环固定值赋值给一个变量上，可以防止循环条件不同步的问题：

|  |
| --- |
| //不推荐方式\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  for (let i = 0; i < arr.length; i++){  console.log(arr.pop())  }  //推荐方式\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  //将循环固定值赋给length变量  let length = arr.length;  for (let i = 0; i < length; i++){  console.log(arr.pop())  } |
|  |

### 异常捕捉

* 当捕捉到异常时，必须处理catch代码区，打印错误信息。原因：不能预料代码运行可能出现的所有情况。为了减少测试周期和避免未知的错误，必须在catch代码区报告错误信息。

|  |
| --- |
| try {  // 代码逻辑  } catch (err) {  //its benefits, I’ll log a message  \_logger.error(err.message)  \_logger.error(err.stack)  } |

* 每个构造器和方法必须明确的声明所有可能抛出的未经制止的异常。原因：为了保持一致性，构造方法和其他方法可以在他们调用的命令上设置约束。文档允许调用者忽略运行期异常。
* 当抛出异常时，不要只提及抛出这个异常的方法的名字，而是使用说明性的文字。原因：报告信息不能只提供方法的名字，一个好的异常信息是非常有用的。
* 确保有一个catch块捕捉和处理所有未能预料的异常。
* 不要用一个未经处理的异常来代替检测异常条件的代码

如：空指针异常可由条件处理

|  |
| --- |
| //不要这样做  let url = jira.url  try{  url = url.trim()  }  catch(err){  logger.error(err.message)  }  //应这样  let url = this.url  url = (!url) ? ‘’ : url.trim(); |

### 变量

* 把一个实例变量暴露在外（这不包括常量的声明）。原因：这是面向对象的原理。使变量为公开的，意味着放弃对类内部结构的控制，而且不能保证变量有有效的值。
* 尽量减少使用变量默认的初始化值。（像String初始化为“”）。原因：尽量减少初始化的错误。
* 尽量减少static。原因：static 变量相当于非oo语言的全局变量。它们使方法依赖于局部的上下文环境，有时会引起同步错误，而且使代码更容易崩溃。
* 用let或者const代替var，这样可以明确变量所处的上下文，且避免了变量提升的现象出现。
* 避免定义多余的存取实例变量的方法。只有真正需要时，才为它们写get/set方法。原因：许多类中的变量的值依赖其它的变量。
* 避免给一个类中的成员变量取和超类相同的名字。原因：它们通常会造成错误。
* 为变量定义名称时，采用小驼峰风格的命名方式。如果为私有变量定义名称，可在变量名头部添加\_符号。
* 在你知道局部变量值时，你才去定义它。原因：尽量减少因为局部变量初始化而带来的问题。
* 宁愿定义和初始化一个局部变量，也不要使用一个碰巧不再使用的一个已经定义的变量。原因：尽量减少局部变量初始化而带来的问题。
* 赋予不再使用的变量的值为null。原因：保证垃圾更容易回收

### 方法

* 写一个仅做“一件事”的方法。
* 尽量将方法体内部的复杂逻辑，拆分成相互独立的方法，以便将来的扩展和维护。
* 方法的参数数量应控制在5个以内，如果参数数量过多，可以用一个对象将所有参数包装起来。
* 方法的名称应按照小驼峰风格命名，如果是私有方法可以在方法名头部添加\_符号。
* 对于箭头函数的使用要注意，但函数体内存在this时，要谨慎使用剪头函数，需要明确this所在的上下文环境。

### 日志

以下内容请使用日志形式记录，不要使用console.log().

* 所有调试信息
* 操作日志
* 系统业务日志
* 系统异常信息

### 其它

* 由于Javascript是弱类型语言，为了消除对方法的入参和返回值的结构的争议，可以在注释信心中加入example，并写明入参或者返回值的接口。例如对象和数组。
* 由于Javascript的运行环境，是一个单线程异步I/O的环境，所以大部分涉及到I/O的方法都是异步的。推荐使用Promise、Generator和async/await语法糖处理I/O的流程，尽量减少回调函数的使用，来避免回调地域的情况出现，保持代码的美观和可读性。
* 不要直接获取某对象内部嵌套对象的内部属性，防止由于嵌套对象为null导致的引用异常。

### 编程技巧

* 异常

顶层的main()函数应该截获所有的异常，并且打印（或者记录在日志中）在屏幕上。

* 数据库连接池

连接完数据库时，必须关闭数据库连接。

* 常量统一管理

为了方便在项目范围内使用各种常量，可以将常量都统一定义在一个文件中，或者同一目录下的多个文件中。

* 统一管理功能类

将jwt校验、文本加密解密、发送邮件等等功能类型的类，统一放入utils目录下管理。便于扩展维护。

* 统一管理响应信息

将http请求之后的响应信息，用一个公用模块进行处理，包含各种情况下的相应处理逻辑。这样，请求处理方法内只需要编写请求处理逻辑即可。

* 枚举类型

Javascript不支持枚举，可以使用暴露常量的方式代替

|  |
| --- |
| const START = 1  const STOP = 2  exports.START = START  exports.STOP = STOP |

这种方式实现了START, STOP 等可以像其他语言的枚举类型一样使用。可以使用“==”操作符来比较。

但是这样使用有一个缺陷：必须要先定义一个常量，才能暴露出去，无法保证此枚举值的准确性。

* 成员函数的可见性

良好的程序设计应该尽可能减小类与类之间耦合，所遵循的经验法则是：尽量限制成员函数的可见性。如果成员函数没必要公有 (public)，就定义为保护 (protected)；没必要保护 (protected)，就定义为私有 (private)。

|  |  |  |
| --- | --- | --- |
| **可见性** | **说明** | **正确用法** |
| public | 公有成员函数可被任何其它对象和类的成员函数调用。 | 当该成员函数必须被该函数所在的层次结构之外的其他对象和类在访问时。 |
| protected | 被保护的成员函数可被它所在的类或该类的子类的任何成员函数调用。 | 当该成员函数提供的行为被它所在类的层次结构内部而非外部需要时。 |
| private | 私有成员函数只可以被该类所在的其它成员函数调用，该类的子类不可以调用。 | 当该成员函数所提供的行为明确针对定义它的类时。私有成员函数常常是重新分配要素的结果。重新分配要素又叫“重组”，指类内其它成员函数封装某一个特定行为的做法。 |

## 代码示例



### Javascript

|  |
| --- |
| 1. /\*\* 2. \* Licensed Materials - Property of tenxcloud.com 3. \* (C) Copyright 2017 TenxCloud. All Rights Reserved. 4. \* v0.1 - 2020-10-22 5. \* @author tianzhiqiang 6. \*/ 7. /\*\* 8. \* 添加 Jira 个人偏好项目设置 9. \* @param {Number} userId 10. \* @param {Object} config 11. \* <code> 12. { 13. project\_type: 'software' 14. project\_key: 'HX', 15. board\_id: 31 16. } 17. \* </code> 18. \*/ 19. exports.createJiraPreference = function\* (userId, config) { 20. config.owner\_id = userId 21. try { 22. yield JiraPreference.insertOnePreference(config) 23. } catch (err) { 24. if (err.name === 'SequelizeUniqueConstraintError'){ 25. return ServiceUtils.responseConflict(err.name) 26. } 27. logger.error(err.message) 28. return ServiceUtils.responseInternalError('try again') 29. } 30. return ServiceUtils.responseSuccess('created') 31. } 32. /\*\* 33. \* 修改Jira 个人偏好项目设置 34. \* @param {Number} userId 35. \* @param {Object} config 36. \* <code> 37. { 38. project\_type: 'software' 39. project\_key: 'HX', 40. board\_id: 31 41. } 42. \* </code> 43. \*/ 44. exports.updateJiraPreference = function\* (userId, config) { 45. const preference = yield JiraPreference.findOneByUserInfo(userId) 46. if (!preference) { 47. return ServiceUtils.responseNotFound(`jira prefrence info not found.`) 48. } 49. delete config.owner\_id 50. try { 51. yield JiraPreference.updateByUserInfo(config, userId) 52. } catch (err) { 53. logger.error(err.message) 54. return ServiceUtils.responseInternalError(err.message) 55. } 56. return ServiceUtils.responseSuccess('updated') 57. } |

# 研发过程



## 研发计划

### 工作描述

项目经理根据项目实际情况，按照项目管理计划模板，制定出项目管理计划。主要包括：项目目标、项目环境资源需求、项目过程定义、主要里程碑、项目人力资源计划、干系人介入计划、项目组培训计划、评审计划、度量数据收集与分析计划、可交付成果清单、方法与工具等。

根据项目管理计划中的里程碑时间点制定项目详细进度计划。进度计划中应进行任务细分，将每个阶段的工作细化，并规定完成时间以及负责人。使用Project编写进度计划。

项目经理需要识别项目潜在风险，并使用《项目风险/重大跟踪表》记录和跟踪风险状态。

配置管理员与项目变更控制委员会（CCB）确认项目的访问控制、基线和版本策略、分支版本策略、构建发布策略、变更控制策略等，建立配置管理计划。

由QA工程师根据项目开发计划中的时间进度，制定针对其具体过程和产品的质量保证计划计划。计划中需要根据项目的特征确定检查哪些过程和工作成果，还需要计划检查的时间和人员。

测试负责人根据项目管理计划和需求规格说明书对项目的测试进度、测试资源的投入、测试的整体思路和测试重点等进行计划，完成项目测试计划。

项目管理计划及附属计划评审，使用评审报告记录和跟踪评审发现问题。如项目工期紧张，该项评审工作可采用邮件评审方式。

需求范围明确后，需要对项目管理计划中的里程碑时间点、人员安排等做调整，各附属计划也进行相应调整。

项目立项且需求基本确定后，由项目经理根据项目的实际情况，会同分管领导意见，提出项目奖金申请，申请过程通过编报《项目详细进度计划》、《项目成本估算、奖金核定表（研发）》、并将完成的相关材料通过邮件发送给运营监管中心项目监管部并抄送主管领导，进行审批。

### 交付工作产品

* 《项目管理计划》
* 《项目详细进度计划》
* 《风险/重大问题跟踪表》
* 《质量保证计划及路线图》

### 文档编写指南

1. **《项目管理计划》**

项目经理在制定项目管理计划时，至少要明确，项目目标与内容、项目所需资源及环境要求、项目过程定义、项目主要里程碑、人力资源计划、干系人介入计划、项目度量计划、项目评审计划、提交的工作产品清单、方法与工具等。

项目过程定义主要是根据项目实际情况对项目过程进行裁剪；项目干系人介入计划主要明确项目相关干系人何时、如何介入项目；项目度量计划和评审计划目前模板中已经有固定的数据了，项目经理可以根据项目情况进行裁剪。需求、开发规模的度量，计划、需求、功能评审都是必须要做的。

1. **《项目详细进度计划》**

根据项目管理计划使用xls或MS Project 编写项目详细计划，项目详细计划中的工作包颗粒度要细，每个工作包需要投入的资源及资源投入百分比都需要填写。项目进度计划制定后，应定期进行跟踪、更新、细化，至少1-2周一次。

1. **《风险/重大问题跟踪表》**

制定项目计划的同时，对项目潜在风险进行识别，并记录在风险/重大问题跟踪表的上半部分，如果风险发生，转变为问题后，应该更新风险的状态，并将问题记录在风险/重大问题跟踪表的下半部分进行跟踪。对风险和问题的跟踪也应是定期的，至少1-2周一次。

1. **《质量保证计划及路线图》**

主要明确质量保证工程师在那些点介入项目，进行审计和检查。

### 经验分享

项目管理计划及附属计划完成后需进行评审，评审可采取邮件或会议形式，为节约资源，一般都建议采取邮件评审方式。项目计划出现变更时，相关附属计划都需要随之进行变更。

项目管理计划中，比较容易忽略的资源环境、干系人介入计划一定要按实际情况填写，评审计划、度量数据收集计划、项目使用的方法与工具应按照项目实际情况进行填写。提交的工作产品清单应尽量详细，命名也要规范，要与项目过程定义时确认需要提交的工作产品保持一致。

制定项目计划阶段要重视风险识别，最好所有干系人都参与风险识别，可以参考类似项目，必要时向专家请教。需求、设计阶段要持续进行风险识别。可以从功能、质量、性能角度识别技术风险，从人力资源状况识别管理风险。

制定项目计划时，选用合适的软件工具，尽量减少项目计划的工作量。

### 方法与工具

在进行进度和成本估算时可使用以下方法：

1. 专家判断法

即请本领域的专家来判断执行项目各工作所需时间的长短。工作持续时间的估计常常是相当困难的，它要涉及到众多的因素，一般很难找到一个通用的计算方法，这个时候专家的历史经验和记录就显得尤为重要，尽管这种方法的结果也具有一定的不确定性和风险，但仍然不失为一种行之有效的方法。

1. 三时估算法

一般地还可以通过估计完成工作的最乐观时间、最悲观时间及最可能时间按下式来估算工作持续时间：工作持续时间 =（乐观的时间＋4×最有可能的时间＋最悲观的时间）/6

1. 经验估算法

进行估计的人应有专门知识和丰富的经验，据此提出一个近似的数字。这种方法是一种最原始的方法，还称不上估算，只是一种近似的猜测。它仅适合于要求很快拿出一个大概数字的项目。

1. 自上而下估算法

此方法一般要求有类似完成项目的经验的情况下使用。其主要内容是：收集上、中层管理人员的经验和判断，以及相关历史数据，然后上、中层管理人员估计整个项目的费用和各个分项目的费用，将此结果传送给下一层管理人员，责成其对组成项目和子项目的任务和子任务的费用进行估算，并继续向下传送其结果，直到项目组的最基层。

这种过程和层级计划制定的过程类似，费用如同项目一样，按照WBS过程，从最高层到最底层进行逐层分解。

这种方法的缺点是特别需要建立好上下管理层畅通的沟通渠道。因为上层管理人员根据经验得出的费用估算结果可能不能满足下层管理人员认为完成任务的需要。此时若不能适当地沟通，费用分配方案失去原有的作用而变成完成项目任务的阻碍，从而导致项目的失败。

使用此方法的好处是中、上层管理人员能够比较准确地掌握项目整体费用分配，从而使项目的费用能够合理地控制在一个水平上，一定程度上避免了项目的费用风险。

1. 自下而上估算法

该方法是指参与项目工作的每一机构和基层单位都估算自己的费用，将估算结果加起来的总和，得到该项目的整个估算费用。具体地可按照WBS体系，从下而上估算各个工作的费用，得到项目的直接费用估计，项目经理再在此基础上加上合理的间接费用，估算出项目的总费用。

根据软件程序模块的标识和定义，估计代码的行数，可近似确定开发和测试软件的费用。

这种方法的缺点在于要保证所有的工作和任务都被考虑到，而且对每个工作单元有过高估算的倾向，往往导致最后的费用估算无法接受。

它的优点在于比起高层管理人员来，底层直接参与项目工作的人员更清楚项目工作所需资源的种类和数量，费用估算更为精确。而且费用估算出自他们自己的估计，可以避免以后费用预算过程中的一些冲突和不满。

1. 类比估算法

即比照以前的经验，或比较以往类似项目的档案资料，根据以前的类似的实际项目的工作时间来推测估计当前项目各工作时间的。如果当前的项目与类比的项目很类似时，类比估计是一种最有效的方法。

类比估算中的不确定性归根结底是由技术人员和费用估算人员所作的主观评价引起的。在多数情况下，可以进行实际的技术比较。问题是在技术差异的基础上建立费用关系，甚至当技术人员做出的所有决策都可以定量客观评价时，费用估算人员还需要确定有关技术发现的费用影响。在很多情况下，这些费用影响是非常主观的，因而，这种估算还是具有较大的不确定性。类比估算适用于项目早期，此时还没有系统的实际费用数据，也没有相似系统的大型数据库，只有此种方法的估算较为准确。

### 自检要素

* 相关文档的命名是否符合公司相关规范，是否以项目编号+项目名称+文档名称的方式命名
* 项目管理计划里程碑时间点是否是具体的日期，并且有标志性提交物
* 项目管理计划最终提交工作产品清单是否全部包含了项目裁剪后生命周期模型所要求的工作产品
* 是否制定了项目组必须的培训计划（包含对内培训和对外培训）
* 是否对项目的风险进行了识别，制定了风险管理措施
* 是否识别了相关干系人，并对干系人介入的活动进行了定义
* 项目进度计划中，工作任务的工期、工时、起止时间、资源是否否已明确定义
* 项目的沟通计划是否清晰明确
* 项目的评审、度量计划是否已制定
* 项目管理计划及其附属计划是否经过正式评审（邮件或会议），并保留了评审记录
* 进行成本估算时，项目的主要需求是否已确定

## 设计与开发

### 工作描述

分析与设计软件的体系结构，产生《概要技术方案说明书》。

对系统数据进行设计，定义各数据实体的存储结构，确定结构、约束、索引、关系等，产生《数据库设计说明书》

设计关键用户界面，设计软件模块的主要接口、类、数据结构、算法，产生《模块设计说明书》

完成代码的编写，并通过单元测试，完成集成测试所需要的产品

### 交付工作产品

* 《技术研究报告》
* 《概要技术方案说明书》
* 《数据库设计说明书》
* 《模块设计说明书》
* 《相关评审报告》

### 文档编写指南

1. **《技术研究报告》**

需要明确项目的技术路线，描述集中可能用到的技术路线，并加以对比分析，最终确认可行的技术路线。

1. **《概要技术方案说明书》**

确定项目的总体框架、关键技术。

1. **《数据库设计说明书》**

主要描述项目总体ER图及各模块的ER图，项目各表之前的关系及表结构。

1. **《模块设计说明书》**

主要对各模块的类图、调用关系、接口等进行设计。

1. **《相关评审报告》**

如项目有设计工作，设计工作产品应及时进行评审，每个项目都必须做功能评审。

### 经验分享

目前公司多数产品都是基于D&A平台进行开发的，一般都是做完需求规格后直接就进行开发，项目的设计工作可以根据项目具体情况进行裁剪，单元测试、功能评审一定要做。

系统设计可分为两个部分进行，首先是总体设计，其任务是系统的总体结构进行设计 ，在此基础上进行第二阶段——详细设计，它的任务是在系统总体设计的指导下，对系统各组成部分进行细致、具体的物理设计，使系统总体设计阶段所作的各种决定具体化 。

对开发人员进行“代码审查、测试、改错”等方面的培训，提高他们的工作效率。

开发小组根据产品的特征，可以适当地修改本规范的各种文档模板。

代码设计的基本原则：具备唯一确定性；标准化与通用性 ；可扩充且易修改；短小精悍即选择最小值代码；具有规律性、便于编码和识别。

### 方法与工具

1. 结构化设计方法

结构化设计方法的基本思想是将系统设计成由多个相对独立，功能单一的模块组成的结构。由于模块之间相对独立，每个模块就可以单独地被理解、编写、测试和修改，从而防止错误在模块间蔓延，提高了系统的质量。它的特点主要有：相对独立，功能单一的模块结构；“块内联系大，块间联系小”的模块性能标准。目前公司的产品大多采用此方法。

一个模块应具有以下四个要素：

1）输入和输出 数据的来源和去向；

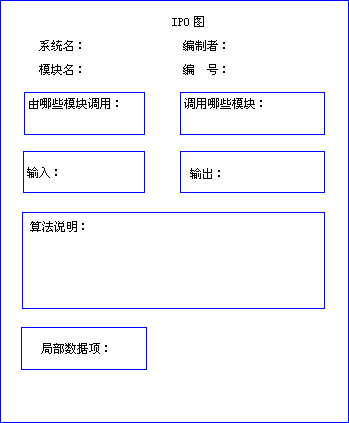
2）处理功能 把输入数据转换为输出数据所做的工作；

3）内部数据 仅供该模块本身引用的数据；

4）程序代码 用来实现模块功能的程序；

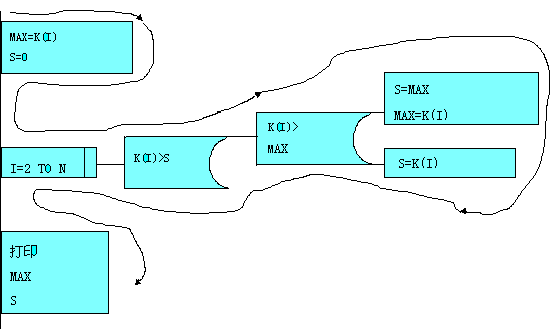
1. IPO图

IPO图是对每个模块进行详细设计的工具。在系统的模块结构图形成过程中，产生大量模块，在详细设计时，开发者为每一个模块写一份说明。用来说明每个模块的输入、输出数据和数据加工。它的主体是算法说明部分，该部分可采用结构化语言，判定表，判定树等。



1. 问题分析图

问题分析图是一种支持结构化程序设计的图形工具，可以用来取代前面所述的程序流程图。 它有逻辑结构清晰、图形标准化等优点， 更重要的是它引导设计人员使用结构化程序设计方法，从而提高了程序的质量。同时，通过比较确定的规则可以由问题分析图直接产生程序，为程序设计的自动化开辟了前景。它是一种由左往右展开的二维树型结构.PAD图的控制流程为自上而下,从左到右地执行.



以上仅参考相关书籍对设计相关的方法和工具作简要介绍，如研发人员需要深入了解可查阅系统设计方法的相关书籍。

### 自检要素

* 文档每一部分的设计是否都可以追溯到需求分析文档或者其他技术文档
* 概要设计说明书的设计目标描述是否明确清晰
* 是否阐述了设计所依赖的运行环境，并与需求分析文档中规定的运行环境相一致
* 是否描述了各类接口的功能、各接口与其他接口或模块之间的关系，接口的设计是否具有可测试性
* 模块的划分是否合适（如：代码规模是否适中？是否便于协同开发？）、模块与模块之间是否具有一定的独立性（如：模块与模块之间应做到低耦合，模块设计应做到高内聚）
* 是否所有的设计可以追踪到需求
* 是否所有计划的模块都得到了详细描述
* 是否每个单元都可以被测试、证明、分析或检查以确定是否符合需求
* 设计文档是否通过评审，且对评审发现问题进行了跟踪和验证
* 编码是否遵循了公司的相关编码规范
* 是否进行了代码审查
* 是否进行了单元测试并保持了相关测试记录
* 是否严格对照需求、功能列表，对项目进行了详细的功能评审

## 发版

### 工作描述

测试通过后，测试人员向配置管理员提供“测试报告”，说明哪个版本测试通过可进行发布，只能由配置管理员进行正式版的编译和发布，打上“正式版”标记，提供给用户使用。

正式发布的版本，配置库中需要作版本标记，同时发布的版本应在指定的路径下进行程序备份。

版本发布后，除在项目管理计划中计划的发版之外（主要指维护期），所有给客户应用的版本（包括临时版）均需由项目经理或配置管理员进行登记，以便于追溯。

### 交付工作产品

* 《发版计划》
* 《发版申请》
* 《发版说明》
* 《相关评审报告》

### 文档编写指南

### 经验分享

产品的发布可以作为一个项目单独立项完成，也可以不立项作为一个发版测试任务完成；

发版测试过程，必须包含实际项目的业务测试，未经项目确认，不算测试通过，不能发布正式版本；

### 自检要素

* 版本发布前是否经过了详细的测试，并获得测试负责人的认可
* 所有缺陷是否都已处理，只剩遗留或者关闭状态缺陷，遗留缺陷是否经过讨论
* 与程序一同发布的文档是否完成并经过评审，如用户手册、版本更新说明等
* 是否达到原定版本发布目标，是否有重要需求遗漏，重要修改未测试
* 发布的版本所有相关工件是否已更新到配置库中，包括源代码、各类文档资料
* 发版程序版本是否备份至产品库，存放路径是否正确
* 发版时是否通过正式邮件通知发版程序版本的访问路径
* 邮件收件人是否包含实施人员、市场人员、测试人员、QA人员以及公司高层经理

## 项目结项

### 工作描述

项目测试、发版完成且已实现项目既定的目标后，由项目经理组织项目组成员对项目配置库进行检查，将未入库的工作产品及时入库。

项目经理撰写项目总结报告，配置管理员梳理项目提交的工作产品清单。

项目组认为项目可以结项后，由项目经理向技术与产品管理委员会、项目研发经理、测试负责人、实施负责人发送项目结项申请邮件，项目总结报告和工作产品清单应随邮件一起发送。

在收到结项申请邮件后，项目高层经理、测试负责人、实施负责人三方对项目进行验证并回复结项意见。主要评价项目目标的达成情况，是否达到结项要求。

信息资产管理部和项目监管部对项目过程检查发现问题的修改情况进行验证，做结项前的工作产品审计和过程审计，对于存在项目管理、配置管理问题的项目应及时纠正，待缺陷修改完成后才能允许项目结项。

项目经理组织召开结项会。

各方验证通过且提交的工作产品确认无误后，由QA工程师发送结项通知，项目正式结项。

项目结项后，需要收集客户或内部满意度调查结果，由QA工程师组织配置管理、测试相关负责人对项目执行情况进行评价。

如申请研发奖励的项目，需要核对项目实际成本，确认项目奖励的实际总盘，填写《项目奖金确认表（研发）》并报相关领导进行审批。

### 交付工作产品

* 《项目总结报告》
* 《项目提交的工作产品清单》
* 《项目评价表》

### 文档编写指南

1. **《项目总结报告》**

主要对项目的投入情况、过程数据（变更、缺陷、工作量等）进行统计。项目经理应该评价一下做整个项目下来的不可量化成果，收集项目经验与教训总结以供其他项目参考。

1. **《项目提交的工作产品清单》**

分类型统计工作产品，需求定义文档放一类、项目管理文档放一类，规模、数量及存放位置也应该尽可能的详细。

1. **《项目评价表》**

从项目管理、产品质量、流程执行情况等方面对项目进行评价，该评价由QA工程师组织配置管理员、测试负责人、实施一同完成。

### 经验分享

项目总结报告是收集项目度量数据的重要来源，在项目开发过程中就应当把各里程碑的工作量、评审情况等数据统计工作放在平时来做，这样不至于结项时写结项报告

项目总结会上可以让项目组成员每人都总结**一下**参与项目过程的收获及建议，这样开结项会才有意义，而不仅仅是为了应付，走个过场，那样很浪费时间和资源。

项目结项时，建议可以做一个系统演示，对项目的功能、性能做一个验证。

### 自检要素

* 测试和发版工作是否已完成
* 是否已完成项目总结报告和提交工作产品清单
* 总结报告是否具备人力资源投入情况、总体工作量等进行分析、总结
* 总结报告是否对项目实施进度进行了总结，计划与实际的偏差以及原因分析、预防措施和解决办法
* 总结报告是否对项目的需求变更、计划变更、人员变动等变更情况进行统计和总结
* 总结报告是否对项目风险和重大问题进行分析、总结
* 项目提交工作产品清单是否全部覆盖项目所产生的所有工作成果（即识别出的全部配置项）
* 是否给技术与产品管理委员会、项目研发经理、实施负责人、测试负责人、配置管理员、QA发送了结项申请邮件
* 项目直接领导、测试、实施人员是否对项目进行了验证
* QA是否对最终工作产品进行了检查
* 配置管理员是否完成基线建立相关文档、并在配置库中建立项目结项/发版基线
* 配置管理员是否按照项目经理或其他人员要求对配置库进行封库操作或做权限调整

# 例行工作



## 例会及报告

### 项目例会

项目例会每周召开一次，由项目经理主持，参与人为项目组成员，不得无故缺席。

会议内容包括：项目进度情况情况，上周工作完成情况，本周工作计划，各项目组成员遇到的问题及需要协调的事宜等。

### 个人周报

项目组成员应在每周五下午18:00之前将个人周报提交至钉钉，发送给项目经理。

个人周报要求内容详实，包括上周工作总结、下周工作计划及问题、风险、建议和感想。

### 项目周报

项目经理应在每周五18:30之前将项目周报提交至钉钉，发送给部门经理。

项目周报要求内容详实，包括项目进度、上周工作总结、下周工作计划及问题和风险。

项目经理对项目周报内容的完整性、数据的准确性及填报时间的及时性负责。

## 研发管理

### 里程碑报告

项目每个里程碑的工作完成后，应编写里程碑报告并在对应的里程碑完成后的一周内提交入库。里程碑报告中需要对整个里程碑的进度进行分析，如进度出现偏差应该分析引起偏差的原因以及纠正偏差的措施。对里程碑的工作量、工作规模，缺陷（含评审发现问题）进行统计。对本里程碑的人员投入、需求变更、风险跟踪、重大问题、培训等活动的情况进行总结。

里程碑报告由项目经理完成，项目组成员负责给项目经理提供素材。如果里程碑报告写好了，将来结项的时候写项目总结就比较容易。

### 项目评审

项目系统设计、功能都需要采用会议方式进行进行评审。

为避免评审会时间过长，且效果不好，建议项目经理将待评审的工作产品提前发给大家预审，尽量要求每个评审、与会人员对工作产品提出1-2个问题，评审会上大家可以针对问题进行讨论，节约评审时间，提高评审效率。需评审的规模、预计花费的时间也应该有个估算，避免长时间开会，开完还没有结论的评审会。

各工作产品负责人带评审结束后应立即统计评审的规模、消耗的时间，评审发现的问题等，这些都可以记录在评审报告、评审问题跟踪表和评审数据汇总表中。评审发现的问题如果解决了，应及时在评审报告或评审问题跟踪表中更新问题的状态，并由第三方对问题的解决情况进行验证。

为使研发、测试、实施三方目标达成一致，测试和实施人员应尽量多参与项目的评审。比较会忽略的计划、用户手册的评审在今后的项目中一定要关注。

### 风险管理

项目经理负责项目总体的风险管理，项目组成员协助项目经理进行风险的处理。

项目经理根据“项目风险/重大问题跟踪表”，定期（例如每周一次）识别本项目的风险。

项目经理组织相关方评估每个风险的严重性、可能性和风险系数，并按照风险系数从高到低的顺序排列风险。对于风险系数较高的，项目经理应当给出风险减缓措施，并指定责任人。风险系数越高，越先处理。

项目经理跟踪风险减缓过程，直到风险已经解决为止。如果风险的性质发生变化，应当及时更新风险减缓措施。如风险已发生，需要将风险转变为问题，并将其放入问题跟踪表的部分进行持续跟踪。

## 项目变更

为了确保重大变更经过评审、确认，保证变更所引起的工作产品能够得到及时、一致的更新并及时入库，保证所有事件性变更有记录，我们需要加强对项目变更的控制。对于严重影响项目进度和成本的需求、设计、人员、计划等的变更必须经过申请、评审、审批方可执行，不允许擅自修改。

### 变更控制的要点

* **变更影响分析**：对每个变更所造成的影响（如：技术影响、成本影响、进度影响等）需进行分析。
* **变更审批**：对于任何已纳入基线的工作产品的更改必须经过审批、并保留审批记录。
* **变更执行与入库**：得到审批的变更及其影响必须得以实施并得到进一步审批方可入库。

### 如何进行变更

* **申请变更**

当项目计划、需求发生重大变更时，项目经理或其他相关负责人提交变更申请（填写《配置项变更控制报告》的“基本信息”和“申请变更”章节的相关内容），需要重点说明“变更内容”和“变更原因”及其“该配置项变更对项目造成的影响”。

典型的变更请求有：项目目标发生改变、增加或减少重大功能需求、技术攻关出现难题、资源调整、进度调整等。

* **评审变更申请**

重点分析此变更对项目造成的影响并回复变更意见。

* **安排变更任务**

同意变更后，项目经理指定变更执行人，对需变更的配置项（受其影响的变更项）执行变更任务，项目经理需要和变更执行人就变更内容达成共识。

* **执行变更任务**

要求文档类的变更必须详细填写变更记录或者变更履历（在每个文档的目录之前），详细到变更内容、变更人、变更时间和变更版本。同时配置管理工具检入时也需要详细输入变更注释；

研发经理负责监督变更任务的执行，如检查变更内容是否正确、是否按时完成工作等；

对于事件性基线变更没有检入检出操作，但需要变更执行人在相关文档中详细填写变更记录，以备后期的变更检查。

注意事项:

* 发生变更时，项目经理必须及时变更执行进度，及变更产生或修改的配置项。
* 对配置项的变更操作应以评审定稿为准、不仅仅以基线为准。即：配置项评审定稿入库后、未建立基线前如对该配置项进行修改，需按照变更流程执行。

# 资料管理



## 文档管理



### 日常资料备份

项目经理要经常对自已负责项目的开发资料进行本机备份并上传至公司文档库中，尤其在项目进行功能升级等改动较大的情况，必须上传至文档库中备份。上传的资料应该标注具体时间、说明，以便于查找、恢复。

每周五或节日放假前一天，所有开发人员必须把最新开发资料上传至文档库中，并由项目经理负责检查。

### 项目资料归档

项目开发工作完成后，由项目经理整理资料，并统一上传至文档库中存档。

## 代码管理

通过GITLIB加强对编码过程的监控，细化工作流程，达到提升软件开发效率，并逐步推进敏捷开发过程，实现代码管理的自动化。

### 岗位划分

1. 代码管理员（Source Code Manager）田鹏

* 负责管理版本管理系统使用者的权限。
* 根据项目新建请求，创建新开发分支并划分权限。

1. 项目开发负责人（Project Leader）田鹏

* 全面负责管理项目所涉及到所有相关资源，包括文档、代码等。
* 审核本项目中所有提交到测试和生产分支上的代码，对其质量和可靠性负有责任。
* 对项目开发进度负责。
* 负责项目开发分支的管理工作。
* 负责监督生产用分支代码的集成/编译/部署。

1. 项目开发组成员（Project Developer）张立群

* 承担具体代码开发工作。
* 负责个人开发分支上代码管理工作。
* 负责个人开发内容的自测工作。
* 对提交到项目分支上的代码质量控制，负有主要责任。

### 版本划分

最新节点应与生产环境中的运行软件保持一致，此分支上的所有节点均满足生产上线要求，并根据实际生产环境代码状态进行演进。完成测试准备上线的项目代码，必须提交到该分支上，进行独立编译生成部署文件。

2. 项目分支

收集开发人员的开发成果，由项目开发负责人统一管理。

此分支的生命周期与项目开发周期一致。项目立项时由代码管理员建立此分支，项目进行中由项目开发负责人管理此分支，所有开发人员的开发成果需要汇总到此分支，项目结束后关闭该分支的提交功能，只允许进行查询。

3. 个人开发分支

由开发组成员自主创建和管理，承担日常开发过程中代码归集，记录详细开发过程。要求每日工作完成必须在该分支上产生节点，每一个功能点均有独立的节点存在。

### 流程说明

1. 立项

代码管理员收到项目成立申请，根据项目归属，从指定的生产分支节点拉出项目分支，将项目组相关人员添加到项目分支下，设定相应权限，提供分支地址等信息给项目经理。

项目经理在项目分支上做初始化设定，做基本修改，建立初始版本后，将项目分支信息提供给开发组成员。

1. 开发

项目组开发成员以项目分支为父分支，建立包含个人姓名的开发子分支（可多个），并在该分支上进行代码修改。在完成修改后，提交代码，在开发环境中获取修改后的代码，进行编译调试和自测，根据调试结果进行后续的代码开发工作。在完成一个功能点的代码开发并自测通过后，将个人开发分支及集成节点信息，提交给测试组成员，进行单个功能点测试。

测试组完成单个功能点测试后，开发成员将个人修改代码和项目分支最新点进行对比，并将对比结果提交给项目经理进行代码评审。项目经理根据评审结果，决定是否将该代码合并到项目分支。

1. 测试

在完成所有的项目开发工作和代码评审后，项目经理将最终的代码节点信息提交项目测试组，由测试组根据节点内容进行编译、部署、测试后，根据测试结果，提交测试报告。

1. 部署

项目经理在项目满足进行生产部署的所有必备条件后，将项目分支的最终测试通过节点，合并到生产分支，并启动生产环境的编译、部署工作。

## 安全管理

开发人员未经允许不得将整个软件或模块的代码相互拷贝，员工之间以文件单元或者相关代码段的方式进行相互交流、学习。GITLIB的代码由专人进行管理，各项目代码访问权限进行严格限制。

开发人员未经允许严禁私自对外发布技术文档。