设计文档

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **文档编号** | | 001 | | **当前版本** | | | 1.0 |
| **当前状态** | | 草稿 |  | **发布日期** | | | 2014/3/31 |
| 发布 | √ |
| **修改历史** | | | | | | | |
| 日期 | 版本 | 作者 | 修改内容 | | 评审号 | 变更控制号 | |
| 2014/3/25 | 1.0 | 宁婧 | 撰写需求规约说明书 | |  |  | |
|  |  |  |  | |  |  | |
|  |  |  |  | |  |  | |

<基于Web的敏捷项目团队协作系统---EasyAgile >

需求规约说明书

**拟制人**： 亓庆国

**审核人：** 亓庆国

**批准人：** 亓庆国

## **目 录**

[**目 录** A-2](#_Toc385598215)

[**1．** **引言** A-3](#_Toc385598216)

[1.1设计依据 A-3](#_Toc385598217)

[1.2假定与约束 A-3](#_Toc385598218)

[**2．** **项目开发约定** A-4](#_Toc385598219)

[2.1常规约定 A-4](#_Toc385598220)

[2.2文件结构 A-4](#_Toc385598221)

[2.2.1文件头信息声明 A-4](#_Toc385598222)

[2.2.2目录结构 A-5](#_Toc385598223)

[2.3命名规则 A-5](#_Toc385598224)

[2.3.1 共性原则 A-5](#_Toc385598225)

[2.4 程序风格 A-6](#_Toc385598226)

[2.4.1空行 A-6](#_Toc385598227)

[2.4.2代码行 A-6](#_Toc385598228)

[2.4.3代码行内的空格 A-7](#_Toc385598229)

[2.4.4对齐 A-8](#_Toc385598230)

[2.4.5长行拆分 A-9](#_Toc385598231)

[2.4.6注释 A-9](#_Toc385598232)

[2.4.7文件头的注释 A-10](#_Toc385598233)

[2.4.8函数头的注释 A-10](#_Toc385598234)

[**3．** **系统架构设计** A-11](#_Toc385598235)

[**4．** **子系统设计** A-12](#_Toc385598236)

[**5．** **系统出错处理设计** A-13](#_Toc385598237)

[**6．** **数据库设计** A-14](#_Toc385598238)

[附录-A. 用例图 A-15](#_Toc385598239)

[附录-B. ERD B-24](#_Toc385598240)

[附录-C. 活动图 C-25](#_Toc385598241)

[附录-D. 附基于web的敏捷项目团队协作系统的相关词汇表 D-29](#_Toc385598242)

## **引言**

本文档为EasyAgile 系统设计说明书。根据系统的需求，我们将在本文档中定义系统的总体架构设计、子系统设计、接口设计、系统出错处理设计以及数据库设计。

本需求规格说明书是为了开发基于Web的敏捷项目团队协作系统---EasyAgile而编写，主要面向系统分析员、程序员、测试员、实施员和最终用户。

本说明书是整个软件开发的依据，它对以后阶段的工作起指导作用，也是项目完成后系统验收的依据。

### 1.1设计依据

* 《EasyAgile\_需求规约说明书.docx》
* 《scope\_and\_vision\_group\_1.doc》

### 1.2假定与约束

* 开发经费方面，由于是课程项目，所以无需资金投入，一切都是在项目组成员课外时间完成。
* 由于是应用服务程序，一切以用户的需求为最根本的出发点。
* 希望开发出的系统应保证界面友好，操作简单明了，性能可靠，易于维护，可扩展，易于升级。
* 开发期限，到6月17日

## **项目开发约定**

为了保证开发过程的顺利进行，在开发过程中每个成员都必须遵守的约定。

在项目团队协作开发的情况下，编程时应该强调的一个重要方面是程序的易读性，在保证软件速度等性能指标能满足用户需求的情况下，能让其他程序员容易读懂你所编写的程序。若研究项目小组的所有开发人员都遵循统一的、鲜明的一套编程风格，可以让协作者、后继者和自己一目了然，在很短的时间内看清楚程序结构，理解设计的思路，大大提高代码的可读性、可重用性、程序健壮性、可移植性、可维护性。

制定本编程规范的目的是为了提高软件开发效率及所开发软件的可维护性，提高软件的质量。本规范由程序风格、命名规范、注释规范、程序健壮性、可移植性、错误处理以及软件的模块化规范等部分组成。

### 2.1常规约定

* **【规则2.1-1】** 小组讨论开会时不要迟到
* **【规则2.1-2】** 每个人都应当尽力完成自己的任务，如果觉得完成不了，一定提前告诉组长
* **【规则2.1-3】** 设计文档按照需求文档编写，代码按照设计文档编写，保证代码的可追溯性
* **【规则2.1-4】** 如果发现设计的缺陷，需及时上报，进行讨论调整。

### 2.2文件结构

本软件项目涉及服务端nodejs编程，前端html编程，以及android app编程。

2.2.1文件头信息声明

* **【规则2.2.1-1】** 对于服务端nodejs与android客户端应用需要在代码文件头加入文件信息声明

|  |
| --- |
| /////////////////////////////////////////////////////////////////  /////////////////////////////////////////////////////////////////  //Copyright(c)2004,DepartmentofMathematics,ZhejiangUniversity  //Allrightsreserved.  //  //Filename：filename.java  //ProjectCode：Theprojectcodeaboutthisfile  //Abstract：Describethecontentofthisfilesummarily  //Reference：......  //  //Version：1.1  //Author：thenameofauthor(mender)  //Accomplisheddate：September2,2004  //  //Replacedversion:1.0  //OriginalAuthor:thenameoforiginalauthor(mender)  //Accomplisheddate:September10,2003  //  //Mainfunctions：  // Function1Returncode Functionname(ParameterExplain)  // Function2Returncode Functionname(ParameterExplain)  // ...  // FunctionnReturncode Functionname(ParameterExplain)  /////////////////////////////////////////////////////////////////  ///////////////////////////////////////////////////////////////// |

示例1-1文件信息声明

* **【规则2.2.1-2】** 文件信息声明以两行斜杠开始，以两行斜杠结束，每一行都以两个斜杠开始；
* **【规则2.2.1-3】** 文件信息声明包含五个部分，各部分之间以一空行间隔；
* **【规则2.2.1-4】** 在主要函数部分描述了文件所包含的主要函数的声明信息，如果没有可以省略

2.2.2目录结构

* **【规则2.2.2-1】** 每个模块占用一个目录
* **【规则2.2.2-2】** 如果某个模块下的文件过多，例如大于20个，则需要进行文件的归类，每一个类占用一个文件夹

### 2.3命名规则

比较著名的命名规则当推“匈牙利”命名法，该命名规则的主要思想是“在变量和函数名中加入前缀以增进人们对程序的理解”。例如所有的字符变量均以ch为前缀，若是指针变量则追加前缀p。如果一个变量由ppch开头，则表明它是指向字符指针的指针。

“匈牙利”法最大的缺点是烦琐，例如

int i,j,k;

float x,y,z;

倘若采用“匈牙利”命名规则，则应当写成

int iI,iJ,ik;//前缀i表示int类型

float fX,fY,fZ;//前缀f表示float类型

如此烦琐的程序会让绝大多数程序员无法忍受。

总的说来，没有一种命名规则可以让所有的程序员赞同，且命名规则对软件产品而言并不是“成败悠关”的事，而且在不同的平台和不同的环境下编写的程序所应遵循的规则也不尽相同，所以我们只是追求制定一种令大多数项目成员满意的命名规则，并在项目中贯彻实施。

2.3.1 共性原则

* **【规则2.3.1-1】** 标识符应当直观且可以拼读，可望文知意，不必进行“解码”；
* **【规则2.3.1-2】** 标识符的长度应当符合“min-length&&max-information”原则；
* **【规则2.3.1-3】** 命名规则尽量与所采用的操作系统或开发工具的风格保持一致；
* **【规则2.3.1-4】** 程序中不要出现仅靠大小写区分的相似的标识符。
* **【规则2.3.1-5】** 程序中不要出现标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但会使人误解；
* **【规则2.3.1-6】** 变量的名字应当使用“名词”或者“形容词＋名词”；
* **【规则2.3.1-7】** 全局函数的名字应当使用“动词”或者“动词＋名词”(动宾词组)；
* **【规则2. 3.1-8】** 用正确的反义词组命名具有互斥意义的变量或相反动作的函数等；
* **【建议2. 3.1-9】** 尽量避免名字中出现数字编号，如Value1,Value2等，除非逻辑上的确需要编号；

**Java开发风格**

### 2.4 程序风格

程序风格虽然不会影响程序的功能，但会影响程序的可读性，追求清晰、美观，是程序风格的重要构成因素。

2.4.1空行

空行起着分隔程序段落的作用。空行得体（不过多也不过少）将使程序的布局更加清晰。空行不会浪费内存，虽然打印含有空行的程序是会多消耗一些纸张，但是值得。

* **【规则2.4.1-1】** 在每个类声明之后、每个函数定义结束之后都要加空行。参见示例3.1(a)；
* **【规则2.4.1-2】** 在一个函数体内，逻揖上密切相关的语句之间不加空行，其它地方应加空行分隔。参见示例3.1(b)；

|  |  |
| --- | --- |
| //blankline  voidFunction1(…)  {  …  }  //blankline  voidFunction2(…)  {  …  }  //blankline  voidFunction3(…)  {  …  } | //blankline  while(condition)  {  statement1;  //blankline  if(condition)  {  statement2;  }  else  {  statement3;  }  //blankline  statement4;  } |

示例**2.4.1** (a)函数之间的空行示例**2.4.1** (b)函数内部的空行

2.4.2代码行

* **【规则2.4.2-1】** 一行代码只做一件事情，如只定义一个变量，或只写一条语句,这样的代码容易阅读，并且方便于写注释；
* **【规则2.4.2-2】** if、for、while、do等语句自占一行，执行语句不得紧跟其后，不论执行语句有多少都要加{}，这样可以防止书写失误；
* **【规则2.4.2-3】** if、for、while、do等语句的“{”要单独占用一行；
* **【建议2.4.2-4】** 所有函数内的变量都在函数开始处定义；
* **【建议2.4.2-5】** 尽可能在定义变量的同时初始化该变量(就近原则)，如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。如果引用了未被初始化的变量，可能会导致程序错误，本建议可以减少隐患。

示例3.2(a)为风格良好的代码行，示例3.2(b)为风格不良的代码行。

|  |  |
| --- | --- |
| intnWidth; //width  intnHeight; //height  intnDepth; //depth | intnWidth,nHight,nDepth;//width,height,depth |
| x=a+b;  y=c+d;  z=e+f; | X＝a+b;y=c+d;z=e+f; |
| if(nWidth<nHight)  {  DoSomething();  } | if(nWidth<nHight)DoSomething(); |
| for(initialization;condition;update)  {  DoSomething();  }  //blankline  Other(); | for(initialization;condition;update)  DoSomething();  Other(); |

示例3.2(a)**风格良好**的代码行示例3.2(b)**风格不良**的代码行

2.4.3代码行内的空格

* **【规则2.4.3-1】** 关键字之后要留空格，象const、virtual、inline、case等关键字之后至少要留一个空格，否则无法辨析关键字，象if、for、while等关键字之后应留一个空格再跟左括号‘（’，以突出关键字；
* **【规则2.4.3-2】** 函数名之后不要留空格，紧跟左括号‘（’，以与关键字区别；
* **【规则2.4.3-3】** ‘（’向后紧跟，‘）’、‘，’、‘;’向前紧跟，紧跟处不留空格；
* **【规则2.4.3-4】** ‘，’之后要留空格，如Function(x,y,z)，如果‘;’不是一行的结束符号，其后要留空格，如for(initialization;condition;update)；
* **【规则2.4.3-5】** 赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=”“>=”、“<=”、“+”、“\*”、“%”、“&&”、“||”、“<<”,“^”等二元操作符的前后应当加空格；
* **【规则2.4.3-6】** 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格；
* **【规则2.4.3-7】** 象“［］”、“.”、“->”这类操作符前后不加空格；
* **【建议2.4.3-8】** 对于表达式比较长的for语句和if语句，为了紧凑起见可以适当地去掉一些空格，如for(i=0;i<10;i++)和if((a<=b)&&(c<=d))

|  |
| --- |
| voidFunc1(intx,inty,intz);//favorablestyle  voidFunc1(intx,inty,intz);//illstyle |
| if(year>=2000)//favorablestyle  if(year>=2000)//illstyle  if((a>=b)&&(c<=d))//favorablestyle  if(a>=b&&c<=d)//illstyle |
| for(i=0;i<10;i++)//favorablestyle  for(i=0;i<10;i++)//illstyle  for(i=0;I<10;i++)//favorablestyle |
| x=a<b?a:b;//favorablestyle  x=a<b?a:b;//illstyle |
| int\*x=&y;//favorablestyle  int\*x=&y;//illstyle |
| array[5]=0;//Donotusearray[5]=0;  a.Function();//Donotusea.Function();  b->Function();//Donotuseb->Function(); |

示例3.3代码行内的空格

2.4.4对齐

* **【规则2.4.4-1】** 程序的分界符‘{’和‘}’应独占一行并且位于同一列，同时与引用它们的语句左对齐；
* **【规则2.4.4-2】** {}之内的代码块在‘{’右边数格处左对齐；
* **【规则2.4.4-3】** 代码的的对齐采用TAB键而不采用空格键对齐，一般TAB键设置为向后空4个空格。

示例3.4(a)为风格良好的对齐，示例3.4(b)为风格不良的对齐。

|  |  |
| --- | --- |
| voidFunction(intx)  {  …//programcode  } | voidFunction(intx){  …//programcode  } |
| if(condition)  {  …//programcode  }  else  {  …//programcode  } | if(condition){  …//programcode  }  else{  …//programcode  } |
| for(initialization;condition;update)  {  …//programcode  } | for(initialization;condition;update){  …//programcode  } |
| While(condition)  {  …//programcode  } | while(condition){  …//programcode  } |
| 如果出现嵌套的｛｝，则使用缩进对齐，如：  {  …  {  …  }  …  } |  |

示例3.4(a)**风格良好**的对齐示例3.4(b)**风格不良**的对齐

2.4.5长行拆分

* **【规则2.4.5-1】** 代码行最大长度宜控制在70至80个字符以内；
* **【规则2.4.5-2】** 长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符），拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

|  |
| --- |
| if((very\_longer\_variable1>=very\_longer\_variable12)  &&(very\_longer\_variable3<=very\_longer\_variable14)  &&(very\_longer\_variable5<=very\_longer\_variable16))  {  DoSomething();  } |
| virtualCMatrixCMultiplyMatrix(CMatrixleftMatrix,  CMatrixrightMatrix); |
| for(very\_longer\_initialization;  very\_longer\_condition;  very\_longer\_update)  {  DoSomething();  } |

示例3.5长行的拆分

2.4.6注释

Js和java语言的注释符为“/\*…\*/”。C++语言中，程序块的注释常采用“/\*…\*/”，行注释一般采用“//…”。注释通常用于：

（1）版本、版权声明；

（2）函数接口说明；

（3）重要的代码行或段落提示。

虽然注释有助于理解代码，但注意不可过多地使用注释。参见示例3.7。

* **【规则2.4.6-1】** 注释是对代码的“提示”，而不是文档，程序中的注释不可喧宾夺主，注释太多了会让人眼花缭乱，注释的花样要少；
* **【规则2.4.6-2】** 如果代码本来就是清楚的，则不必加注释；例如

i++; //i加1，多余的注释

* **【规则2.4.6-3】** 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性，不再有用的注释要删除；
* **【规则2.4.6-4】** 注释应当准确、易懂，防止注释有二义性，错误的注释不但无益反而有害；
* **【规则2.4.6-5】** 尽量避免在注释中使用缩写，特别是不常用缩写；
* **【规则2.4.6-6】** 注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，不可放在下方；
* **【规则2.4.6-8】** 当代码比较长，特别是有多重嵌套时，应当在一些段落的结束处加注释，便于阅读；
* **【建议2.4.6-9】** 对于多行代码的注释，尽量不采用“/\*...\*/”，而采用多行“//”注释，这样虽然麻烦，但是在做屏蔽调试时不用查找配对的“/\*...\*/”。

|  |  |
| --- | --- |
| ////////////////////////////////////////////////////////////////////  //Functioncapacity：  //Parameterdeclare：  //Returnvalue：  ////////////////////////////////////////////////////////////////////  voidFunction(floatx,floaty,floatz)  {  …  } | if(…)  {  …  while(…)  {  …  }//endofwhile  …  }//endofif |

示例3.7程序的注释

2.4.7文件头的注释

文件头的注释请参见1.1，文件头的注释是以两行斜杠开始，以两行斜杠结束(以区别于函数的注释)。

2.4.8函数头的注释

一般说来每个函数都应该做详细的注释，函数头的注释是以一行斜杠开始，以一行斜杠结束，注释的内容包括“功能”，“参数”，“返回值”，“设计思想”，“调用函数”，“日期”，“修改记录”等几个方面，函数头的注释格式如下：

|  |
| --- |
| //////////////////////////////////////////////////////////////////////////////////////////////  //Functioncapacity： *Describethefunctioncapacity*  //Parameterdeclare：  // *parameter1：* *Describethefunctionofparameter(input/outputparameter)*  // *parameter2： Describethefunctionofparameter(input/outputparameter)*  // *......*  //Returnvalue：*Describethepossiblereturnvalue*  //Designedidea：*Describedesignedideaaboutthefunction*  //Author：  //Creationdate：*Creationdate(YY-MM-DD)*  //Transferredfunction： *Listthesub-functioninthefunction*  //Modificationrecord：  // (一)Mender1： Modifieddate： modifiedcontent  ////////////////////////////////////////////////////////////////////////////////////////////// |

## **系统架构设计**

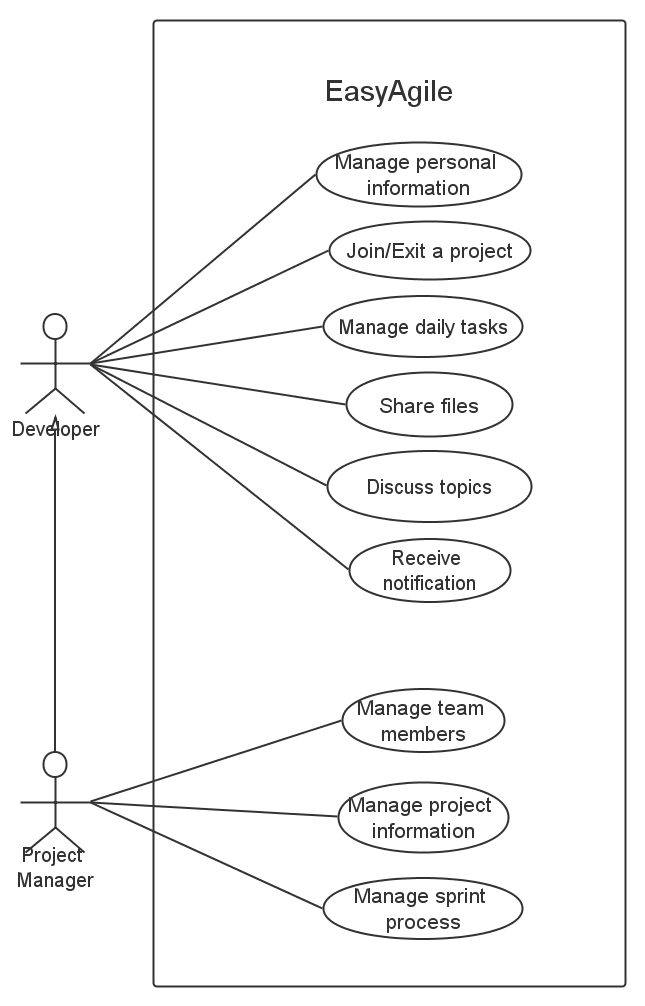
## **子系统设计**

## **系统出错处理设计**

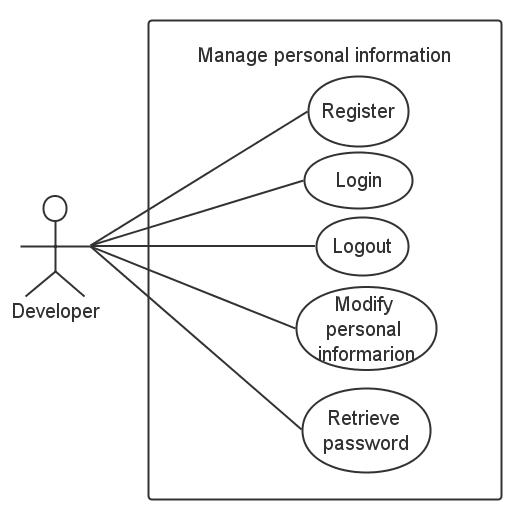
## **数据库设计**

# 用例图

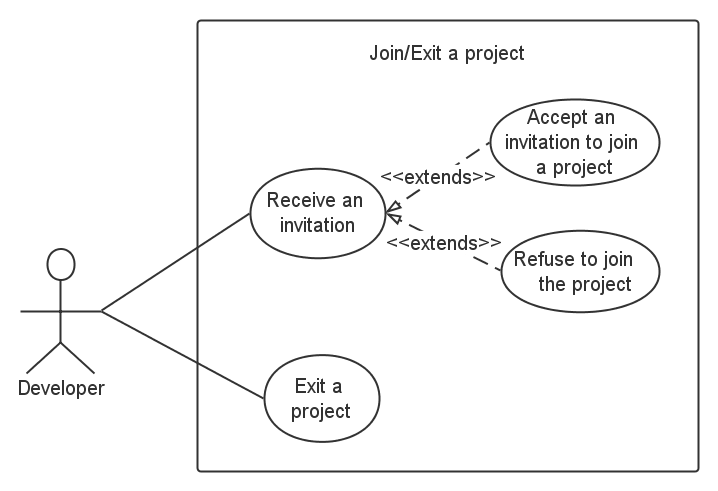
* 1. **基于web的敏捷项目团队系统总用例图概览**



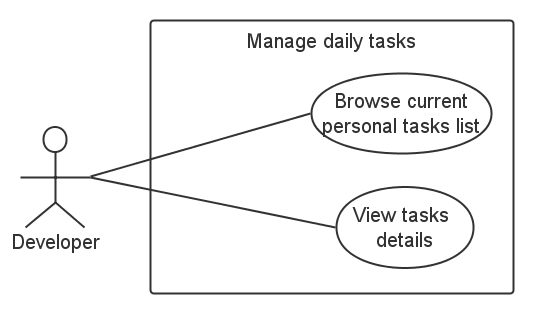
* 1. **用户个人管理模块**

****

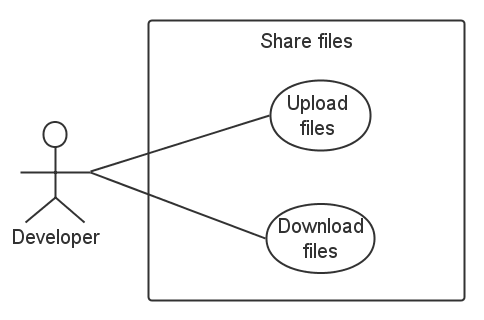
* 1. **加入/退出项目组模块**

****

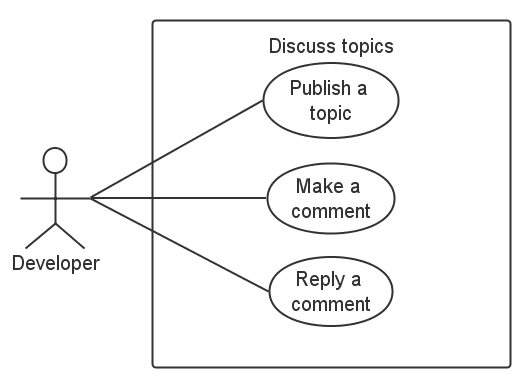
* 1. **管理个人任务模块**



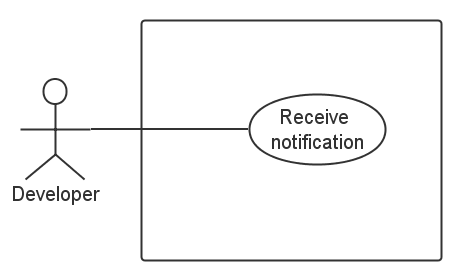
* 1. **文件共享模块**



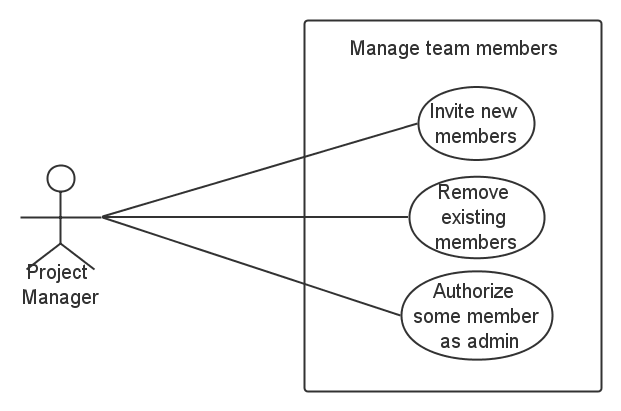
* 1. **话题讨论模块**

****

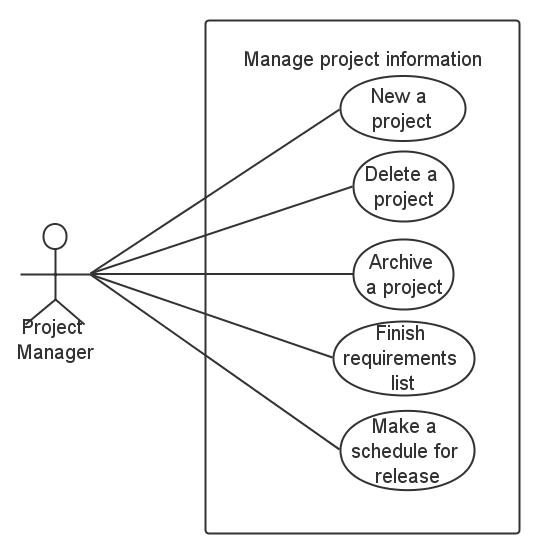
* 1. **接收消息通知**



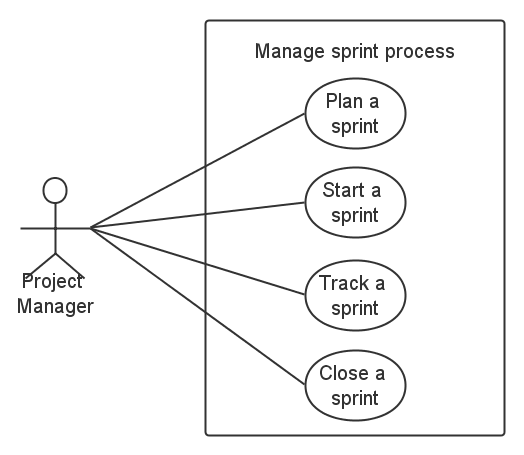
* 1. **团队成员管理模块**



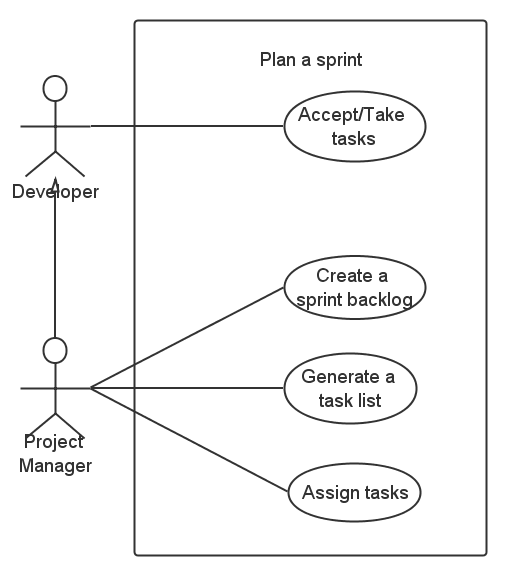
* 1. **项目基本信息管理模块**



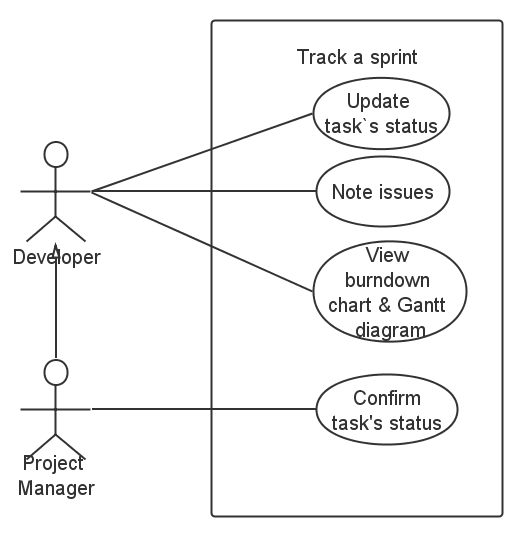
* 1. **迭代过程管理模块**



* 1. **迭代计划模块**

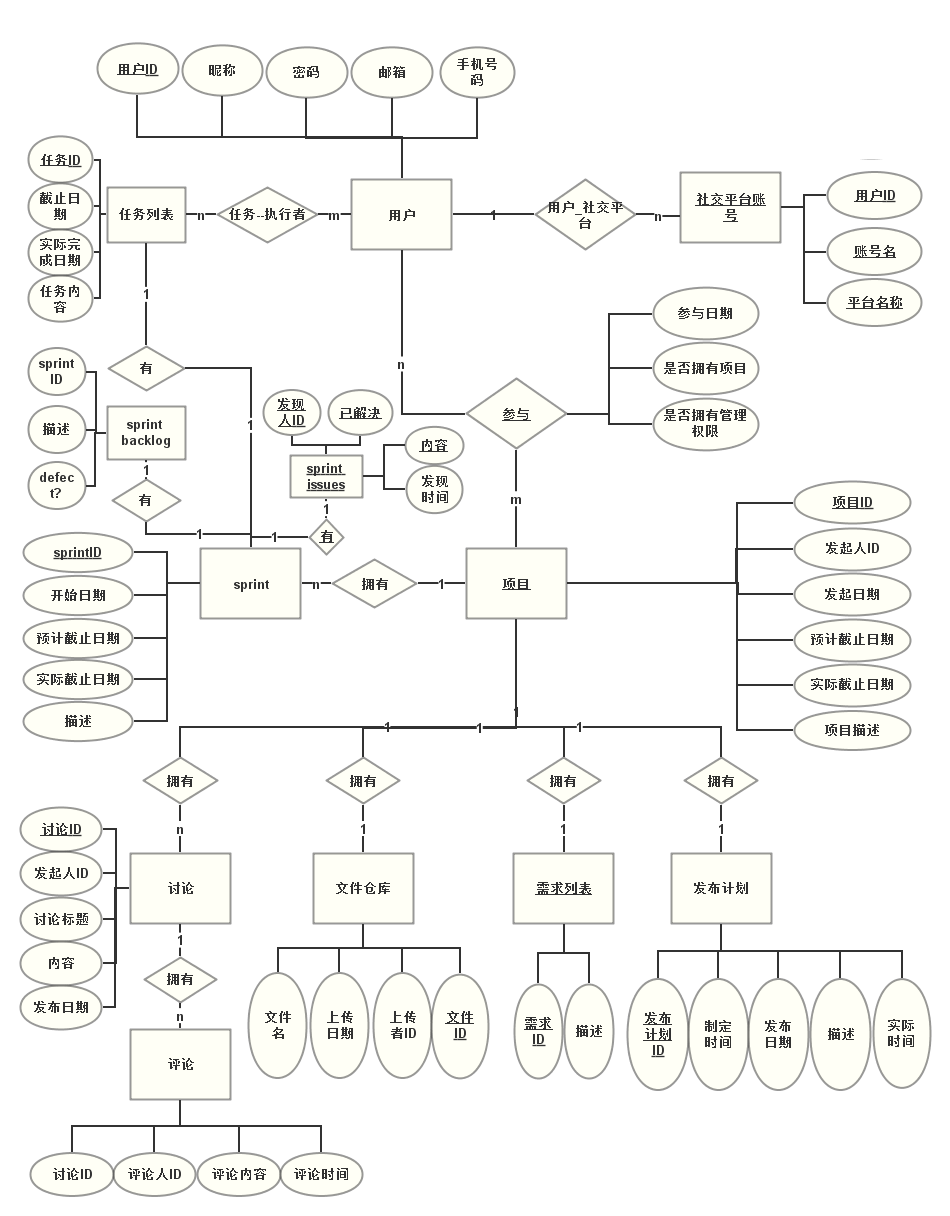


* 1. **迭代追踪模块**



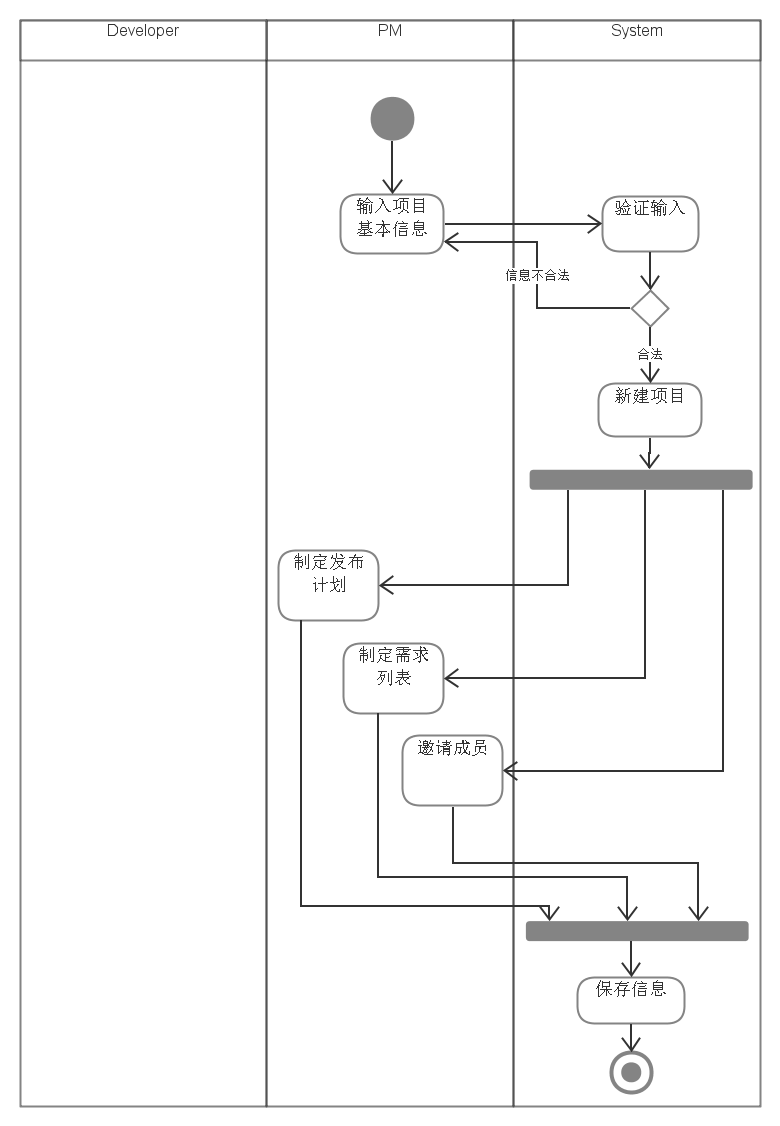
# ERD

* 1. **系统ERD**

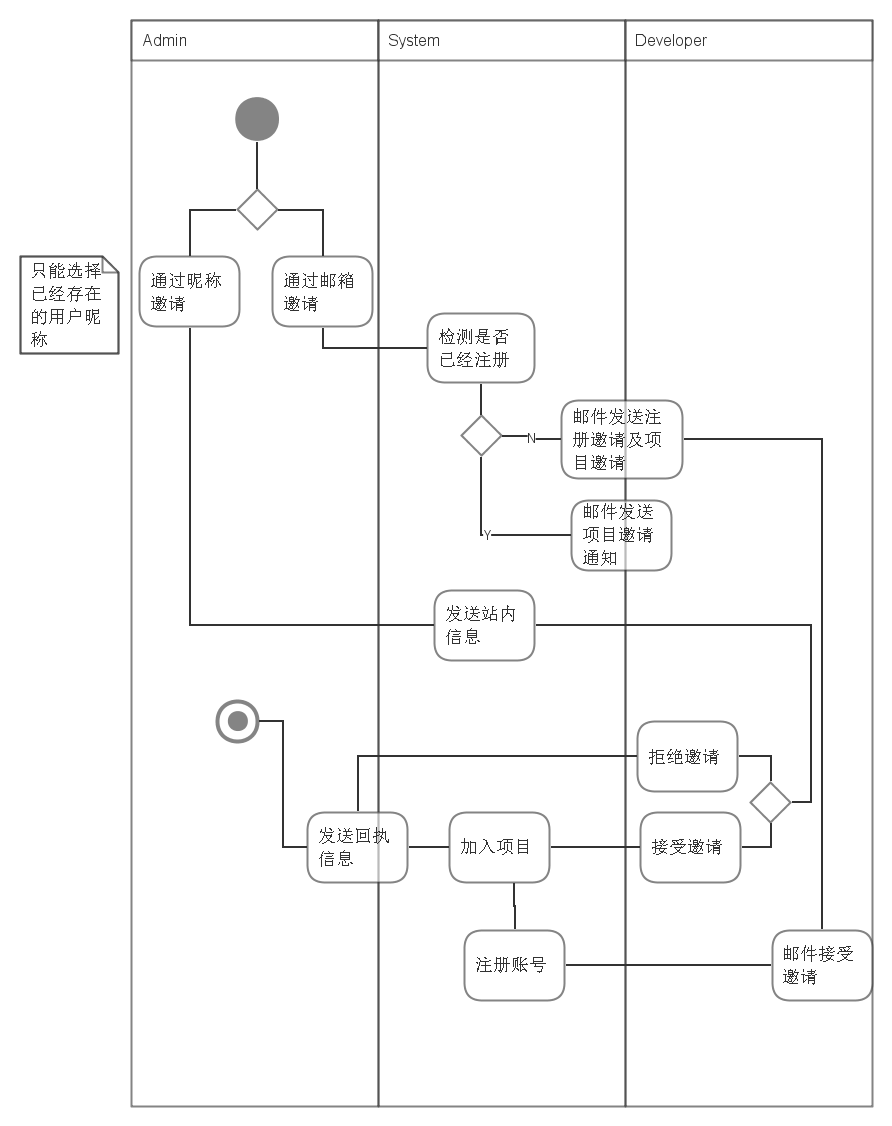


# 活动图

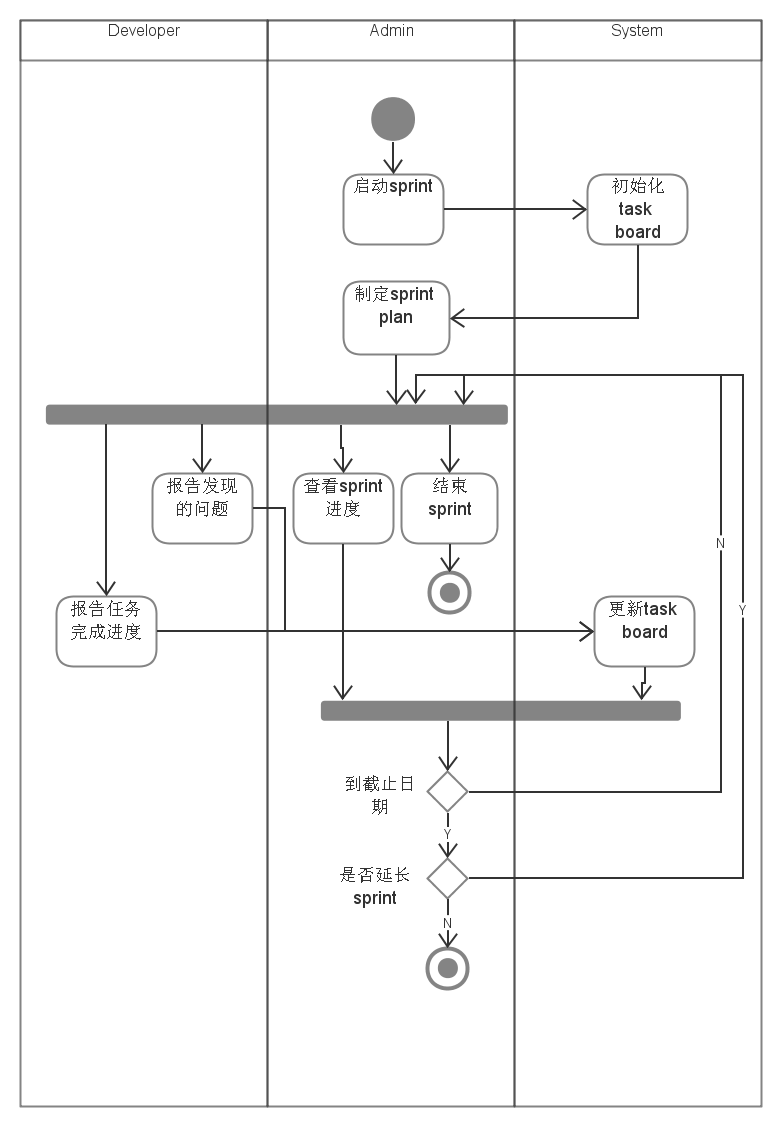
* 1. **新建项目活动图**



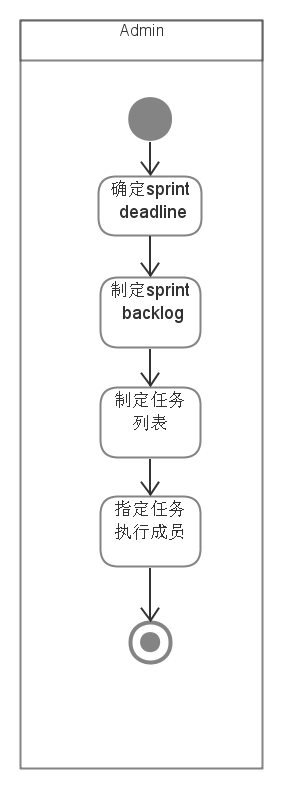
* 1. **邀请团队成员活动图**



* 1. **管理sprint活动图**



* 1. **计划sprint活动图**



# 附基于web的敏捷项目团队协作系统的相关词汇表

|  |  |
| --- | --- |
| 术语 | 定义 |
| 用户 | **在没有特殊的修饰词修饰的情况下，用户一般指已注册的使用该网站的个人；新用户指的是未注册的潜在使用者** |
|  |  |
| PM | **拥有最高权限的特殊用户，特殊权限包括查看所有项目成员的信息，以及编写需求列表和sprint backlog，指派任务给其他成员，移除现有的成员等** |
|  |  |
| 项目管理员 | **项目指的是用户在系统中自己创建的项目，这里的项目管理员指的是整个项目中各个子系统的负责人，类似于部门经理** |
|  |  |
| 归档项目 | 被归档的项目可以是未完成而且不需要继续完成的，也可以是已经完成的，归档后不能再对项目信息进行任何形式的编辑，只允许查看 |
|  |  |
| assigned to | **sprint backlog中任务的执行者、认领者或是完成者** |
|  |  |
| priority | **sprint backlog中任务的优先级，拟用1-10的数字来表示，数字越大，优先级越高** |
|  |  |
| time estimated | **sprint backlog中任务的估时，将会由团队在线下通过扑克牌游戏得出，单位是“人时”，表示一个人完成该任务需要花费的小时数** |
|  |  |