基于Git配置管理工具的可视化动态表单生成器

目录

[一、 软件项目管理 3](#_Toc500710193)

[1.1软件项目管理的重要性 3](#_Toc500710194)

[1.1.1什么是项目管理？ 3](#_Toc500710195)

[1.1.2为什么要有项目管理？ 3](#_Toc500710196)

[1.1.3项目管理在软件开发中应用的成因 3](#_Toc500710197)

[1.1.4软件项目管理常见问题及解决方案 3](#_Toc500710198)

[1.2软件配置管理工具的发展历史及功能分析 3](#_Toc500710199)

[1.2.1关于软件配置管理 3](#_Toc500710200)

[1.2.2软件配置管理工具的起源 3](#_Toc500710201)

[1.2.3软件配置管理工具的变迁 4](#_Toc500710202)

[1.2.4软件配置管理工具的主要功能 4](#_Toc500710203)

[1.2.5软件配置管理工具的工作流 4](#_Toc500710204)

[1.3对本项目中使用的配置管理工具Git的分析 5](#_Toc500710205)

[1.3.1Git版本控制管理工具的简介 5](#_Toc500710206)

[1.3.2Git的工作原理 6](#_Toc500710207)

[1.3.3Git的基本使用方法 8](#_Toc500710208)

[1.3.4Git的分支 10](#_Toc500710209)

[1.3.5Git的分布式 10](#_Toc500710210)

[二、 项目选题 10](#_Toc500710211)

[2.1项目选题 11](#_Toc500710212)

[2.2项目目标 11](#_Toc500710213)

[2.3基本设计概念 11](#_Toc500710214)

[2.4对功能的规定 11](#_Toc500710215)

[2.5功能需求与用户的关系 11](#_Toc500710216)

[2.6业务处理流程 11](#_Toc500710217)

[2.7结构设计 13](#_Toc500710218)

[2.7.1结构设计原则： 13](#_Toc500710219)

[2.7.2该系统的接口设计： 14](#_Toc500710220)

[2.7.3该系统的数据库设计： 15](#_Toc500710221)

[2.8对系统架构的规定 16](#_Toc500710222)

[2.8.1系统架构原则： 16](#_Toc500710223)

[2.8.2系统架构设计： 16](#_Toc500710224)

[2.9前端界面设计 19](#_Toc500710225)

[三、 可视化动态表单生成器的开发过程管理 22](#_Toc500710226)

[3.1开发环境 22](#_Toc500710227)

[3.1.1服务器 22](#_Toc500710228)

[3.1.2集成开发工具 22](#_Toc500710229)

[3.2开发管理过程 22](#_Toc500710230)

[3.2.1项目工程建立与版本库初始化 22](#_Toc500710231)

[3.2.2项目管理标准 25](#_Toc500710232)

[3.2.3定期例会 25](#_Toc500710233)

[3.2.4Git使用规范说明 26](#_Toc500710234)

[3.2.5版本命名规则 26](#_Toc500710235)

[3.2.6开发流程 26](#_Toc500710236)

[3.2.7编码命名规则 28](#_Toc500710237)

[四、 项目管理中配置管理工具的应用 30](#_Toc500710238)

[4.1配置管理工具GIT的安装与初始化 30](#_Toc500710239)

[4.2创建版本库 31](#_Toc500710240)

[4.3建立远程仓库 34](#_Toc500710241)

[五、 开发质量控制 37](#_Toc500710242)

[5.2设计阶段 38](#_Toc500710243)

[5.3编码阶段 38](#_Toc500710244)

[5.4测试阶段 39](#_Toc500710245)

[5.6软件质量保证SQA中的7个方面的要求详细解释 39](#_Toc500710246)

[（1）定义恰当的质量目标 39](#_Toc500710247)

[（2）让每个人都知道质量的重要性 39](#_Toc500710248)

[（3）调整团队个人的目标，纳入质量考核体系 39](#_Toc500710249)

[（4）获取正确的需求 40](#_Toc500710250)

[（5）将测试重点放在最关键和风险很高的点 40](#_Toc500710251)

[（6）提高设计质量 40](#_Toc500710252)

[（7）合理使用自动化测试工具 40](#_Toc500710253)

[六、 开发效率保证 40](#_Toc500710254)

[6.1保证开发效率的意义 40](#_Toc500710255)

[6.2软件开发中的心态 41](#_Toc500710256)

[6.3提高开发效率 41](#_Toc500710257)

[6.4降低维护开销 42](#_Toc500710258)

[七、 总结与展望 43](#_Toc500710259)

# 软件项目管理

## 1.1软件项目管理的重要性

### 1.1.1什么是项目管理？

项目管理是指把各种系统、方法和人员结合在一起，在规定的时间、预算和质量目标范围内完成项目的各项工作。即从项目的投资决策开始到项目结束的全过程进行计划、组织、指挥、协调、控制和评价，以实现项目的目标。

所谓管理包含领导、组织、用人、计划、控制等五项主要工作。

### 1.1.2为什么要有项目管理？

按照传统的做法，当企业设定了一个[项目](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE)后，参与这个[项目](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE)的至少会有好几个部门，包括[财务部门](http://wiki.mbalib.com/wiki/%E8%B4%A2%E5%8A%A1%E9%83%A8%E9%97%A8)、市场部门、行政部门等等，而不同部门在运作项目过程中不可避免地会产生摩擦，须进行[协调](http://wiki.mbalib.com/wiki/%E5%8D%8F%E8%B0%83)，而这些无疑会增加项目的[成本](http://wiki.mbalib.com/wiki/%E6%88%90%E6%9C%AC)，影响项目实施的[效率](http://wiki.mbalib.com/wiki/%E6%95%88%E7%8E%87)。

而项目管理的做法则不同。不同职能部门的成员因为某一个项目而组成[团队](http://wiki.mbalib.com/wiki/%E5%9B%A2%E9%98%9F)，[项目经理](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE%E7%BB%8F%E7%90%86)则是[项目团队](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE%E5%9B%A2%E9%98%9F)的[领导者](http://wiki.mbalib.com/wiki/%E9%A2%86%E5%AF%BC%E8%80%85)，他们所肩负的责任就是[领导](http://wiki.mbalib.com/wiki/%E9%A2%86%E5%AF%BC)他的团队准时、优质地完成全部工作，在不超出[预算](http://wiki.mbalib.com/wiki/%E9%A2%84%E7%AE%97)的情况下实现[项目目标](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE%E7%9B%AE%E6%A0%87)。项目的管理者不仅仅是项目执行者，他参与项目的需求确定、[项目选择](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE%E9%80%89%E6%8B%A9)、计划直至收尾的全过程，并在时间、成本、[质量](http://wiki.mbalib.com/wiki/%E8%B4%A8%E9%87%8F)、[风险](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9)、[合同](http://wiki.mbalib.com/wiki/%E5%90%88%E5%90%8C)、[采购](http://wiki.mbalib.com/wiki/%E9%87%87%E8%B4%AD)、[人力资源](http://wiki.mbalib.com/wiki/%E4%BA%BA%E5%8A%9B%E8%B5%84%E6%BA%90)等各个方面对项目进行全方位的管理，因此项目管理可以帮助企业处理需要跨领域解决的复杂问题，并实现更高的运营效率。

### 1.1.3项目管理在软件开发中应用的成因

项目管理的应用从80年代仅限于建筑、国防、航天等行业迅速发展到今天的[计算机](http://wiki.mbalib.com/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA)、电子[通讯](http://wiki.mbalib.com/wiki/%E9%80%9A%E8%AE%AF)、[金融业](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E4%B8%9A)甚至政府机关等众多领域。

人们通常认为，项目管理是第二次世界大战的产物（如：曼哈顿计划）。在1950年至1980年期间，应用项目管理的主要是国防建设部门和建筑公司。

### 1.1.4软件项目管理常见问题及解决方案

在有限的时间、资金内，要满足不断增长的软件产品质量要求；开发的环境日益复杂，[代码共享](https://baike.baidu.com/item/%E4%BB%A3%E7%A0%81%E5%85%B1%E4%BA%AB)日益困难，需跨越的平台增多；程序的规模越来越大；软件的重用性需要提高；软件的维护越来越困难。

## 1.2软件配置管理工具的发展历史及功能分析

### 1.2.1关于软件配置管理

[软件配置管理](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E9%85%8D%E7%BD%AE%E7%AE%A1%E7%90%86)是一种标识、组织和控制修改的技术。[软件配置](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E9%85%8D%E7%BD%AE)管理应用于整个[软件工程过程](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B%E8%BF%87%E7%A8%8B)。在软件建立时变更是不可避免的，而变更加剧了项目中软件开发者之间的混乱。SCM活动的目标就是为了标识变更、控制变更、确保变更正确实现并向其他有关人员报告变更。从某种角度讲，SCM是一种标识、组织和控制修改的技术，目的是使错误降为最小并最有效地提高生产效率。

### 1.2.2软件配置管理工具的起源

[能力成熟度集成模型](http://wiki.mbalib.com/w/index.php?title=%E8%83%BD%E5%8A%9B%E6%88%90%E7%86%9F%E5%BA%A6%E9%9B%86%E6%88%90%E6%A8%A1%E5%9E%8B&action=edit)是由[美国卡耐基·梅隆](http://wiki.mbalib.com/wiki/%E7%BE%8E%E5%9B%BD%E5%8D%A1%E8%80%90%E5%9F%BA%C2%B7%E6%A2%85%E9%9A%86)大学的软件工程研究所[组织开发](http://wiki.mbalib.com/wiki/%E7%BB%84%E7%BB%87%E5%BC%80%E5%8F%91)，并于2002年发布的一种规范、实用的途径来管理软件过程的模型。[CMMI](http://wiki.mbalib.com/wiki/CMMI)通过指导软件开发人员的活动来改进[软件过程](http://wiki.mbalib.com/wiki/%E8%BD%AF%E4%BB%B6%E8%BF%87%E7%A8%8B)，以达到软件过程可复用性、可[定量管理](http://wiki.mbalib.com/wiki/%E5%AE%9A%E9%87%8F%E7%AE%A1%E7%90%86)、可[有效控制](http://wiki.mbalib.com/wiki/%E6%9C%89%E6%95%88%E6%8E%A7%E5%88%B6)的目的。软件配置管理是[CMMI](http://wiki.mbalib.com/wiki/CMMI)可重复级的一个[关键过程域](http://wiki.mbalib.com/wiki/%E5%85%B3%E9%94%AE%E8%BF%87%E7%A8%8B%E5%9F%9F)，其目的是在整个项目的软件生命周期中，保持软件产品的完整性和可追踪性，这包含了对改变的控制和所有能影响到改变的软件因素的管理。作为过程实现、过程优化的一部分，配置管理是实现软件过程的基本保证，它还是基于重用的软件开发的管理手段，所以成为软件[过程管理](http://wiki.mbalib.com/wiki/%E8%BF%87%E7%A8%8B%E7%AE%A1%E7%90%86)的核心。CMMI模型清晰地描述了SCM，并说明了SCM 的目的和所要达到的目标，具体描述了某级成熟度下软件过程在该方面所应达到的一组目标和实现这些目标的一组关键实践。这些关键实践被划分为5类，分别为完成该组目标所需的承诺、执行能力、执行的活动、度量分析以及验证。使企业在实施软件配置管理时能知道到底要做什么，团队的配置管理现状如何评估，在哪些方面还可以进行改进等问题能得到具体的答案。

### 1.2.3软件配置管理工具的变迁

随着[软件工程](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B)的发展，软件配置管理越来越成熟，从最初的仅仅实现[版本控制](https://baike.baidu.com/item/%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6)，发展到21世纪初提供[工作空间](https://baike.baidu.com/item/%E5%B7%A5%E4%BD%9C%E7%A9%BA%E9%97%B4)管理、并行开发支持、过程管理、权限控制、变更管理等一系列全面的管理能力，已经形成了一个完整的理论体系。同时在软件配置管理的工具方面，也出现了大批的产品，如：最著名的ClearCase；有将近二十年历史的[Perforce](https://baike.baidu.com/item/Perforce)；开源产品CVS；入门级工具Microsoft VSS；新秀Hansky Firefly。

### 1.2.4软件配置管理工具的主要功能

针对市场的这些需求，Hansky公司在中国市场推出了业界技术领先的软件配置管理解决方案，产品包括配置管理工具Firefly和变更管理工具Butterfly。Firefly是Hansky公司推出的软件配置管理系统，它可以轻松管理、维护整个企业的软件、代码和文档。Firefly是一个高性能、运行速度极快的软件配置管理系统，支持不同的开发、运行平台，因此它能在整个企业中的不同团队、不同项目中都得以广泛的应用。Firefly能够对团队开发提供有力的支持，开发团队一旦拥有了Firefly，就可以非常准确的定义：软件将在什么时间发布；当前发布版本中有哪些功能，由哪些组件构成；当前版本中加入了针对哪些Bug的修改；软件的某个修改是谁认可的；如何建立新的发布版本等等。

### 1.2.5软件配置管理工具的工作流

1）规划、调整网络开发环境

一个规划良好的开发环境，是实施配置管理系统的前提。此阶段要对配置管理系统做出规划，主要考虑以下问题：

网络的[带宽](http://wiki.mbalib.com/wiki/%E5%B8%A6%E5%AE%BD)、拓扑结构

服务器的选择、命名规范

存储区的定位

开发人员及组的命名规约等。

2）设计配置管理库

根据项目开发的要求，设计开发资源的存储模式，良好的存储模式有利于减轻管理上的负担，增强配置管理库的访问性能，同时便于控制访问权限，保护软件资产。

3）定义配置管理系统的角色

需要确定与配置管理相关的所有角色，包括他们的相应的活动。在开发过程中，一个开发人员可能兼任多种角色，但一项任务在同一时刻只能由一个角色来执行。

一般配置管理中的角色主要包括：

[项目经理](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE%E7%BB%8F%E7%90%86)

配置管理员

软件开发人员

集成人员

QA人员

4）制定配置管理流程

配置管理实施的一个重要阶段，主要目的是根据项目开发的需要，制定相应的配置管理流程，以更好地支持开发，主要活动包括：

定制并行开发策略。合理的并行开发策略应该具有以下特点：协调项目的复杂性和需求，统一创建分支类型和[元数据](http://wiki.mbalib.com/wiki/%E5%85%83%E6%95%B0%E6%8D%AE)，为开发过程中的变更集成制定有效的规范，适时反映开发过程中方法和需求的变化：

发布版本管理。软件开发过程中的一个关键活动是提取工件的相关版本，以形成软件系统的阶段版本或发布版本，一般将其称为稳定基线。一个稳定基线代表新开发活动的开始，而一系列定制良好的活动之后又会产生一个新的稳定基线。有效地利用此项功能，在项目开发过程中可以至始至终管理、跟踪部件版本间的关联。

5）相关人员的培训

实施配置管理系统，相关人员需要接受培训：

管理员培训：针对配置管理员，主要学习配置管理工具管理相关内容：

开发人员培训：针对开发人员，主要学习配置管理工具与开发相关的常用操作

管理流程培训：针对全体人员，目的是了解配置管理策略和流程，以及如何与开发管理、[项目管理](http://wiki.mbalib.com/wiki/%E9%A1%B9%E7%9B%AE%E7%AE%A1%E7%90%86)相结合。

## 1.3对本项目中使用的配置管理工具Git的分析

### 1.3.1Git版本控制管理工具的简介

Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。

Git是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

Git --- The stupid content tracker, 傻瓜内容跟踪器。Linus Torvalds 是这样给我们介绍 Git 的。

Git 是用于 Linux内核开发的版本控制工具。与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持（wingeddevil注：这得分是用什么样的服务端，使用http协议或者git协议等不太一样。并且在push和pull的时候和服务器端还是有交互的。），使源代码的发布和交流极其方便。 Git 的速度很快，这对于诸如 Linux kernel 这样的大项目来说自然很重要。 Git 最为出色的是它的合并跟踪（merge tracing）能力。

实际上内核开发团队决定开始开发和使用 Git 来作为内核开发的版本控制系统的时候，世界开源社群的反对声音不少，最大的理由是 Git 太艰涩难懂，从 Git 的内部工作机制来说，的确是这样。但是随着开发的深入，Git 的正常使用都由一些友好的脚本命令来执行，使 Git 变得非常好用，即使是用来管理我们自己的开发项目，Git 都是一个友好，有力的工具。现在，越来越多的著名项目采用 Git 来管理项目开发.

作为开源自由原教旨主义项目，Git 没有对版本库的浏览和修改做任何的权限限制。

目前GIT已经可以在windows下使用，主要方法有二：msysgit和Cygwin。Cygwin和Linux使用方法类似，Windows版本的GIT提供了友好的GUI(图形界面)，安装后很快可以上手，不在此做大篇幅介绍。

### 1.3.2Git的工作原理

**直接记录快照，而非差异比较**

Git 和其他版本控制系统的主要差别在于，Git 只关心文件数据的整体是否发生变化，而大多数其他系统则只关心文件内容的具体差异。这类系统（CVS，Subversion，Perforce，Bazaar 等等）每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容：



图 ‑1 其他系统在每个版本中记录着各个文件的具体差异

Git 并不保存这些前后变化的差异数据。实际上，Git 更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。为提高性能，若文件没有变化，Git 不会再次保存，而只对上次保存的快照作一链接。Git 的工作方式就如下图所示。



图 一‑2 Git 保存每次更新时的文件快照

这是 Git 同其他系统的重要区别。它完全颠覆了传统版本控制的套路，并对各个环节的实现方式作了新的设计。Git 更像是个小型的文件系统，但它同时还提供了许多以此为基础的超强工具，而不只是一个简单的 VCS。稍后会讨论 Git 分支管理的时候，我们会再看

看这样的设计究竟会带来哪些好处。

**近乎所有操作都是本地执行**

在 Git 中的绝大多数操作都只需要访问本地文件和资源，不用连网。但如果用 CVCS 的话，差不多所有操作都需要连接网络。因为 Git 在本地磁盘上就保存着所有当前项目的历史更新，所以处理起来速度飞快。

举个例子，如果要浏览项目的历史更新摘要，Git 不用跑到外面的服务器上去取数据回来，而直接从本地数据库读取后展示给你看。所以任何时候你都可以马上翻阅，无需等待。如果想要看当前版本的文件和一个月前的版本之间有何差异，Git 会取出一个月前的快照和当前文件作一次差异运算，而不用请求远程服务器来做这件事，或是把老版本的文件拉到本地来作比较。

用 CVCS 的话，没有网络或者断开 VPN 你就无法做任何事情。但用 Git 的话，就算你在飞机或者火车上，都可以非常愉快地频繁提交更新，等到了有网络的时候再上传到远程仓库。同样，在回家的路上，不用连接 VPN 你也可以继续工作。换作其他版本控制系统，这么做几乎不可能，抑或非常麻烦。比如 Perforce，如果不连到服务器，几乎什么都做不了（默认无法发出命令 p4 edit file 开始编辑文件，因为 Perforce 需要联网通知系统声明该文件正在被谁修订。但实际上手工修改文件权限可以绕过这个限制，只是完成后还是无法提交更新。）；如果是 Subversion 或 CVS，虽然可以编辑文件，但无法提交更新，因为数据库在网络上。看上去好像这些都不是什么大问题，但实际体验过之后，你就会惊喜地发现，这其实是会带来很大不同的。

**时刻保持数据完整性**

在保存到 Git 之前，所有数据都要进行内容的校验和（checksum）计算，并将此结果作为数据的唯一标识和索引。换句话说，不可能在你修改了文件或目录之后，Git 一无所知。这项特性作为 Git 的设计哲学，建在整体架构的最底层。所以如果文件在传输时变得不完整，或者磁盘损坏导致文件数据缺失，Git 都能立即察觉。

Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个 SHA-1 哈希值，作为指纹字符串。该字串由 40 个十六进制字符（0-9 及 a-f）组成。

Git 的工作完全依赖于这类指纹字串，所以你会经常看到这样的哈希值。实际上，所有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名。

**多数操作仅添加数据**

常用的 Git 操作大多仅仅是把数据添加到数据库。因为任何一种不可逆的操作，比如删除数据，都会使回退或重现历史版本变得困难重重。在别的 VCS 中，若还未提交更新，就有可能丢失或者混淆一些修改的内容，但在 Git 里，一旦提交快照之后就完全不用担心丢失数据，特别是养成定期推送到其他仓库的习惯的话。

这种高可靠性令我们的开发工作安心不少，尽管去做各种试验性的尝试好了，再怎样也不会弄丢数据。至于 Git 内部究竟是如何保存和恢复数据的，我们会在后面讨论 Git 内部原理时再作详述。

**文件的三种状态**

对于任何一个文件，在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。

由此我们看到 Git 管理项目时，文件流转的三个工作区域：Git 的工作目录，暂存区域，以及本地仓库。



图 ‑3 工作目录，暂存区域，以及本地仓库

每个项目都有一个 Git 目录（如果 git clone 出来的话，就是其中 .git 的目录；如果 git clone --bare 的话，新建的目录本身就是 Git 目录。），它是 Git 用来保存元数据和对象数据库的地方。该目录非常重要，每次克隆镜像仓库的时候，实际拷贝的就是这个目录里面的数据。

从项目中取出某个版本的所有文件和目录，用以开始后续工作的叫做[工作目录](http://www.nowamagic.net/academy/tag/%E5%B7%A5%E4%BD%9C%E7%9B%AE%E5%BD%95)。这些文件实际上都是从 Git 目录中的压缩对象数据库中提取出来的，接下来就可以在工作目录中对这些文件进行编辑。

所谓的暂存区域只不过是个简单的文件，一般都放在 Git 目录中。有时候人们会把这个文件叫做索引文件，不过标准说法还是叫暂存区域。

### 1.3.3Git的基本使用方法

Git 支持许多数据传输协议，包括本地传输、 git://协议、http(s):// 或者 SSH传输协议 user@server:/path.git，除了HTTP协议之外，其他所有协议都要求在服务器端安装并运行Git。

1）使用本地仓库

对于本地的项目管理，一些具体的操作方法可以参考这篇文章：[Git基础](http://www.open-open.com/lib/view/open1328069733264.html) 里面详细的讲解了Git每个操作的使用方法和效果。因为本文主要目的在于演示如何利用GitHub参与开源项目，对于使用细节就不再花时间描述了，不过我认为仔细的看看这些使用方法对于提高工作效率非常有帮助。

使用本地仓库，用户只是自己一个人，所以不存在协同工作的问题，不管怎么玩儿，一般不会出问题。使用远程仓库，情况就会复杂、有趣的多。

2）使用远程仓库

要参与任何一个 Git 项目的协作，必须要了解该如何管理远程仓库。远程仓库是指托管在网络上的项目仓库，可能会有好多个，其中有些你只能读，另外有些可以写。同他人协作开发某 个项目时，需要管理这些远程仓库，以便推送或拉取数据，分享各自的工作进展。管理远程仓库的工作，包括添加远程库，移除废弃的远程库，管理各式远程库分 支，定义是否跟踪这些分支，等等。

以现在比较流行的GitHub为例，如果我在上面创建了一个项目，实际上相当于使用 git init 新建了一个服务器端的仓库。如果我想在本地进行开发，那么我就需要 git clone 到我的本地。做了一些开发之后，我可以 git push 将本地的修改推送到服务器仓库中。随着项目发展，有其他人想要参与到这个项目中来，他可以在GitHub上Fork我这个项目，这样他对这个项目才有写权限，而且可以将他的工作保存到GitHub的服务器上。如果他希望将自己的工作提交给我，首先他需要在本地开发环境中添加我的远程仓库 git remote add。然后 git push remotename master 发起推送的请求，如果我接受了，他的工作就可以合并到主干中了。因为我们此时是并行开发，如果他想看我的工作，可以采用 git pull remotename 的方式将我所做的修改拉取到本地，非常的方便。

上面这段话，描述了我们在使用远程仓库以及与其他人协作过程中的大体流程，需要用到的一些远程仓库的操作如下：

1）查看当前配置的远程仓库

可以使用 git remote -v 来查看当前项目中都添加了哪些远程仓库

其中Origin一般是自己在服务器上的远程仓库，其他的为他人的远程仓库。

2）添加新的远程仓库

要添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用，运行 git remote add [shortname] [url]

git remote add git@github.com：DianaLuna/visible-dynamic-form-builder.git

3）抓取远程仓库的信息

git fetch [remote-name]

此命令会到远程仓库中拉取所有你本地仓库中还没有的数据。运行完成后，你就可以在本地访问该远程仓库中的所有分支，将其中某个分支合并到本地，或者只是取出某个分支，一探究竟。如果是克隆了一个仓库，此命令会自动将远程仓库归于 origin 名下。所以，git fetch origin 会抓取从你上次克隆以来别人上传到此远程仓库中的所有更新（或是上次 fetch 以来别人提交的更新）。有一点很重要，需要记住，fetch 命令只是将远端的数据拉到本地仓库，并不自动合并到当前工作分支，只有当你确实准备好了，才能手工合并。

4） 从远程仓库抓取信息并合并

git pull [remote-name]

可以使用 git pull 命令自动抓取数据下来，然后将远端分支自动合并到本地仓库中当前分支。在日常工作中我们经常这么用，既快且好。实际上，默认情况下git clone 命令本质上就是自动创建了本地的 master 分支用于跟踪远程仓库中的 master 分支（假设远程仓库确实有 master 分支）。所以一般我们运行git pull，目的都是要从原始克隆的远端仓库中抓取数据后，合并到工作目录中的当前分支。

5）推送数据到远程仓库

git push [remote-name] [branch-name]

项目进行到一个阶段，要同别人分享目前的成果，可以将本地仓库中的数据推送到远程仓库。实现这个任务的命令很简单： git push [remote-name] [branch-name]。如果要把本地的 master 分支推送到origin 服务器上（再次说明下，克隆操作会自动使用默认的 master 和 origin 名字），可以运行下面的命令:

git push origin master

只有在所克隆的服务器上有写权限，或者同一时刻没有其他人在推数据，这条命令才会如期完成任务。如果在你推数据前，已经有其他人推送了若干更新，那 你的推送操作就会被驳回。你必须先把他们的更新抓取到本地，合并到自己的项目中，然后才可以再次推送。

6）查看远程仓库信息

git remote show [remote-name]

7）远程仓库的删除和重命名

git remote rename ［old name] [new name]

git remote rm [remote-name]

### 1.3.4Git的分支

Git 中的分支，其实本质上仅仅是个指向 commit 对象的可变指针。Git 会使用 master 作为分支的默认名字。在若干次提交后，就有了一个指向最后一次提交对象的 master 分支，它在每次提交的时候都会自动向前移动。

### 1.3.5Git的分布式

每一台计算机都是服务器。Git可以建立树状的结构，每一台Clone了库的计算机都包含完整的数据。我们可以以该库为服务器，继续进行下载。Git服务器可以在本地进行提交，不需要联网。只需要在网络畅通时push一次即可。一旦服务器挂掉，可以立刻找其他电脑充当服务器。

# 项目选题

## 2.1项目选题

## 2.2项目目标

随着各行各业的快速发展，表单的使用变得更加的广泛和频繁。然而对于一些中小型企业来讲，并没有足够的人力物力和技术支持来保证随时能进行表单的数据库修改，比如一个旅游公司需要大量的调查问卷数据，在第一次租用技术人员构造完数据库后对表单的需求产生了变化，就需要再一次地租用技术人员，产生不必要的开销。由此，本可视化动态表单设计器旨在能以简洁明了的用户界面操作来进行表单数据库的修改。

## 2.3基本设计概念

产品总体效果：能够实现表单的动态化管理，不需要修改后台代码就能对表单属性进行增删查改（进阶：UI模块修改，表单形式修改）。

软件设计包括：软件的结构设计、数据设计，接口设计和过程设计。

结构设计：定义软件系统各主要部件之间的关系。

数据设计：将模型转换成数据结构的定义。

接口设计：软件内部，软件和操作系统间以及软件和人之间如何通信。

过程设计：系统结构部件转换成软件的过程描述。

## 2.4对功能的规定

产品功能：表单制作、表单修改、用户填写等。

该系统分为四大模块：登陆模块、注册模块、管理修改模块、使用模块

登陆模块：用户名输入、密码输入、身份选择、身份信息验证、注册选择

管理修改模块：表单增删查改、退出登录功能、操作记录、表单查找

注册模块：用户名输入、密码输入、录入系统

登陆系统：用户名输入

## 2.5功能需求与用户的关系

用户特征：缺乏系统后台代码修改能力的非软件公司普通职员。

## 2.6业务处理流程

1）登录模块：

用户/管理员进入登陆界面

输入用户名

输入密码

选择身份

点击登陆

流程描述：用户在进入登录界面后，首先输入用户名和密码进行身份的确认，登录完成之后进行身份的选择，进入对应的界面。

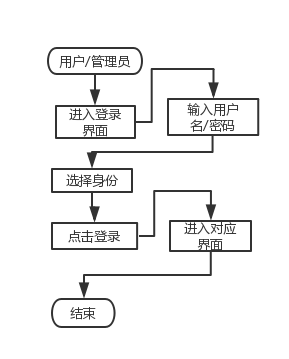


图 二‑1

2）注册模块：

没有用户名的用户点击“注册”进入注册模块

用户名输入

密码输入

注册： 在点击注册后将用户信息录入数据库之中

流程描述：游客在访问网站之后，若对相关部分感兴趣，则可选择注册成为会员，进入注册界面，填入自己的用户名和密码，点击注册，系统自动在后台进行查重，若不存在相同账户，则注册成功。

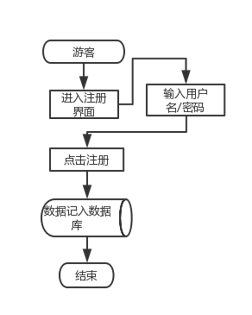


图 二‑2

3）管理修改系统模块：

管理员登录之后进入表单编辑模式

表单增删查改 : 管理员可以使用不同按钮选择对表单的增删查改。

退出登录功能：管理员点击按钮退出登录到登陆界面

操作记录：表单编辑模式右侧显示管理员所操作的每一步 //退出登录后清空

表单查询：管理员可以通过用户id查找到其填写的表单

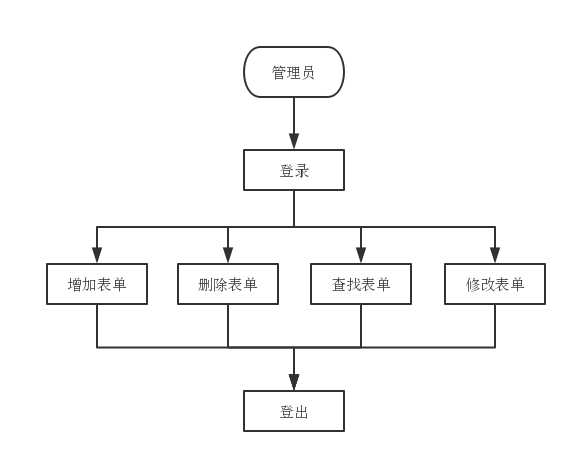


图 二‑3

4）使用模块：

填写表单

通过按键上传表单

## 2.7结构设计

### 2.7.1结构设计原则：

软件结构设计是把一个软件需求变换为软件表示的过程，是对软件的整体结构、程序结构、是对软件的整体结构、程序结构、数据结构、文件结构、接口定义等的设计，是宏观上的设计。

软件结构设计的目标是综合采用各种技术手段，将系统需求转换为数据结构、模块结构（或对象/类结构）的表达形式，并实现系统的性能、安全性、可靠性要求。取得最佳方案。

最佳方案的标准如下：

1. 开发费用低；
2. 资源消耗低；
3. 开发时间短；
4. 生产效率高；
5. 可靠性较高；
6. 可维护性好；

结构设计是一个自顶向下，逐步具体的综合性的设计过程。主要内容有：

1. 综合分析系统的各种实现方案，提出最佳实现方案的建议；
2. 指定设计规范、标准及约定原则；
3. 软件结构设计：采用某种软件设计方法，按照模块的构造原则，逐步、逐层地设计软件的模块层次结构；
4. 数据结构以及接口的设计
5. 系统性能设计
6. 系统安全性能设计：系统的自保护设计，数据一致性设计，容错设计；
7. 系统可靠性设计；
8. 设计文档的编写：结构设计说明书，用户手册，系统初步的测试计划说明书；
9. 设计的审查和复审

### 2.7.2该系统的接口设计：

* 1. 登陆界面；

|  |  |  |  |
| --- | --- | --- | --- |
| **接口名称：UserLoginService** | | | |
| **接口名** | **功能说明** | **参数说明** | **返回值说明** |
| findUserTree | 根据用户输入的账号，匹配用户头像，身份等相关信息 | 用户ID | 返回用户信息集合 |
| MatchUserInfo | 根据返回信息进行匹配，若密码正确则登入 | 输入的密码，用户信息集合 | 返回一个布尔值，指示登录状态 |

* 1. 注册界面；

|  |  |  |  |
| --- | --- | --- | --- |
| **接口名称：UserRegisterService** | | | |
| **接口名** | **功能说明** | **参数说明** | **返回值说明** |
| newUser | 在数据库中新建一个账户 | 用户ID，用户密码，用户头像，用户身份 | 返回创建信息 |

* 1. 不同人员以不同身份登陆之后分别进入 使用界面/管理修改界面；

|  |  |  |  |
| --- | --- | --- | --- |
| **接口名称：UserManageService** | | | |
| **接口名** | **功能说明** | **参数说明** | **返回值说明** |
| newTable | 在用户数据库下新建一个表格 | 用户ID，表格名称，字段信息（列） | 返回一个表ID指向数据库索引 |
| newColumn | 在表中新建一个字段 | 表ID，字段ID，字段属性，字段字数限制 | 返回该表 |
| deleteColumn | 从表中删除一个字段 | 表ID，字段ID | 返回该表 |
| openTable | 打开用户文件下已有的某个表格 | 用户ID，表ID | 返回一个布尔值表示执行状态 |

### 2.7.3该系统的数据库设计：

数据库需求：

虽然各种表单之间具有较大的差异性，但它们可以总结出一些共性：

（1） 所有表单都是由表头、表中、表尾三部分组成的；

（2） 表头一般是由描述性的信息组成的，表尾是由总结性或补充性的信息组成的；

（3） 表中是表单信息量最大的一部分，它可能由不同的信息单元组成，但是不同的信息单元最终是由含有业务数据的单元格组成的；

（4） 所有的实例表都是在只具有表单内在属性的样式单的基础上，填写了具体的业务数据后生成的；

为了满足客户不断的需求变化，动态表单要能够快速地增删查改其中的属性，这样的需求以往普通的数据库设计便无法实现，我们不能使用事先确定好且无法修改的字段。

a） Form\_用户表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Form\_User(Form\_用户表)** | | | | | | | | |
| **是否主键** | **字段名** | **字段描述** | **数据类型** | **长度** | **可空** | **约束** | **缺省值** | **备注** |
| 是 | userId | 用户名 | varchar(10) |  | 否 |  |  |  |
|  | password | 密码 | number(6) |  | 否 |  |  |  |
|  | level | 身份 | int |  | 否 |  |  |  |
|  | description | 描述 | varchar(100) |  | 是 |  |  |  |

b） Form\_控件表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Form\_Widget(Form\_控件表)** | | | | | | | | |
| **是否主键** | **字段名** | **字段描述** | **数据类型** | **长度** | **可空** | **约束** | **缺省值** | **备注** |
| 是 | widget\_name | 控件名 | varchar(10) |  | 否 |  |  | 单行文本；多行文本；日期；单选； |
|  | widget\_default | 默认值 | varchar |  | 是 |  |  |  |

c） Form\_配置表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Form\_Configure(Form\_配置表)** | | | | | | | | |
| **是否主键** | **字段名** | **字段描述** | **数据类型** | **长度** | **可空** | **约束** | **缺省值** | **备注** |
| 是 | configure\_name | 表单名 | varchar(10) |  | 否 |  |  | 用来显示，如姓名，性别 |
|  | configure\_default | 字段默认值 | varchar |  | 是 |  |  |  |
|  | configure\_type | 控件类型 | varchar |  | 否 |  |  |  |
|  | configure\_length | 控件长度 | int |  | 否 |  |  |  |
|  | configure\_describe | 字段描述 | varchar(100) |  | 否 |  |  |  |
|  | judge | 是否必填 | int |  | 否 |  |  |  |

d） Form\_管理员表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Form\_User(Form\_用户表)** | | | | | | | | |
| **是否主键** | **字段名** | **字段描述** | **数据类型** | **长度** | **可空** | **约束** | **缺省值** | **备注** |
| 是 | administrator\_Id | 管理员名 | varchar(10) |  | 否 |  |  |  |
|  | password | 密码 | number(6) |  | 否 |  |  |  |
|  | level | 身份 | int |  | 否 |  |  |  |

## 2.8对系统架构的规定

### 2.8.1系统架构原则：

动态应用，是相对于网络静态内容而言的，是指以c/c++、php、Java、perl、.net等服务器端语言开发的网络应用软件，比如论坛、网络相册、交友、BLOG等常见应用。动态应用系统通常与数据库系统、缓存系统、分布式存储系统等密不可分。

大型动态应用系统平台主要是针对于大流量、高并发网站建立的底层系统架构。大型网站的运行需要一个可靠、安全、可扩展、易维护的应用系统平台做为支撑，以保证网站应用的平稳运行。

大型动态应用系统又可分为几个子系统：

1）Web前端系统

2）负载均衡系统

3）数据库集群系统

4）缓存系统

5）分布式存储系统

6）分布式服务器管理系统

7）代码分发系统

### 2.8.2系统架构设计：

* **系统层次结构**

使用模块

管理修改

登录模块

注册模块

图 二‑4

业务逻辑层

表单生成

操作读取

数据修改

图 二‑5

数据库修改

数据库生成

图 二‑6

* **应用层主体结构图**

可视化动态表单设计器

登录模块

注册模块

使用模块

管理修改模块

图 二‑7

* **登录模块**

登录模块

用户名输入

密码输入

身份选择

身份信息验证

注册选择

图 ‑8

* **管理修改模块**

管理修改模块

表单增删查改

退出登陆功能

操作记录

表单查找

图 二‑9

* **注册模块**

注册模块

用户名输入

密码输入

确认写入数据库

核实已注册用户

图 二‑10

* **使用模块**

使用模块

表单填写修改

上传功能

图 二‑11

## 2.9前端界面设计

首先是登陆界面：



图 二‑12

在上图中，用户可以查看基本的图表设计，若身份为管理员，则可进行登录创建相应的表单。

这是管理员的登陆界面：

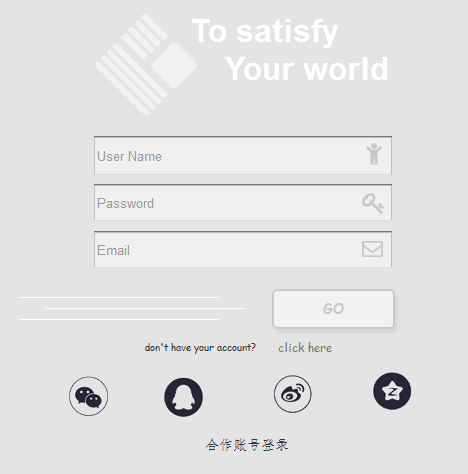


图 二‑13

这里在用户输入了自己的身份账号，网页使用了Ajax异步同步技术，即时显示已注册用户的头像和相关名字。

登录完成后打开编辑页面：

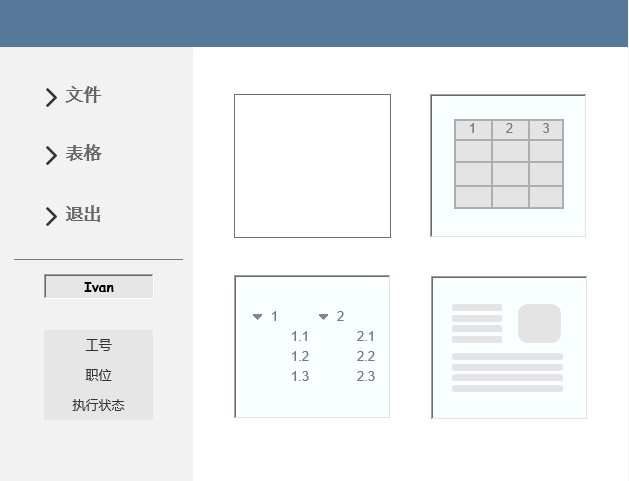


图 二‑14

这里可以在右边选取预设的几个模块进行编辑操作。

左侧的三个菜单栏可以添加了动作捕捉，鼠标悬浮即可打开列表项

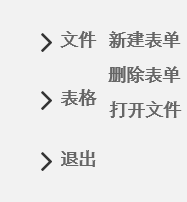


图 二‑15

选择新建表单之后会打开如下的添加界面：



图 二‑16

在该界面中用户可以添加相关的表单字段

# 可视化动态表单生成器的开发过程管理

## 3.1开发环境

### 3.1.1服务器

Oracle JDK 1.8

Apache Tomcat 8.0.26以上版本

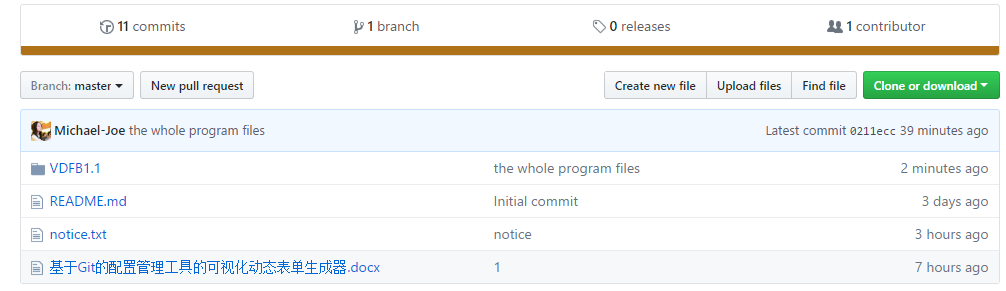
### 3.1.2集成开发工具

Eclipse、MyEclipse

## 3.2开发管理过程

### 3.2.1项目工程建立与版本库初始化

1）在Git服务器建立项目



2）在本地建立工程项目，项目目录为VisibleDynamicFormBuilder

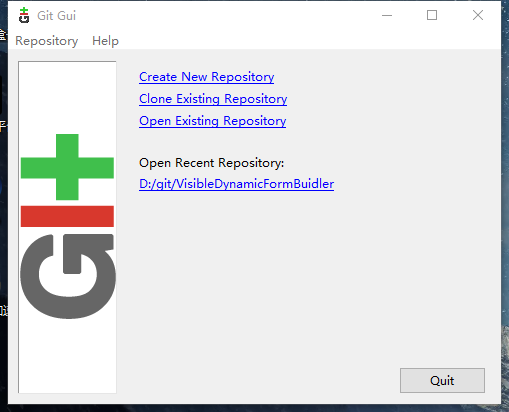


图 ‑1本地工程项目创建

3）VisibleDynamicFormBuilder库初始化

a）打开Git Bash，进入命令行界面

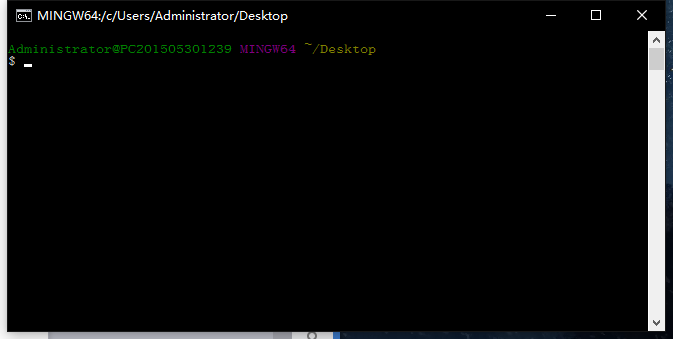


图 ‑2进入Git命令行界面

b）进入项目建立的目录MINGW64 / d/git/VisibleDynamicFormBuilder

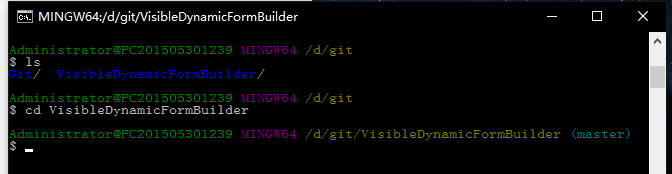


图 ‑3进入项目管理目录

c）初始化Git,生成.git文件

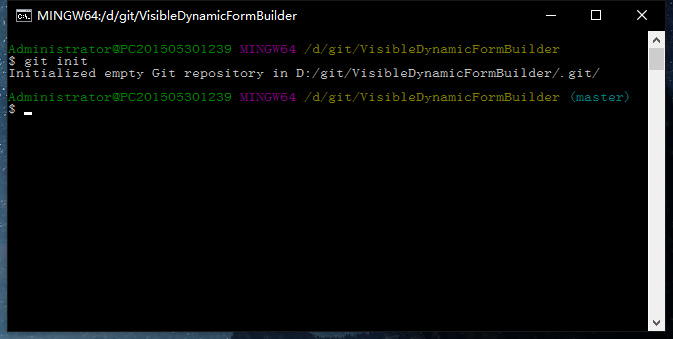


图 ‑4Git init命令行输入

d）初始化完毕



图 ‑5初始化文件

e）建立branch分支

创建分支1

图 ‑6建立项目分支

同理，建立分支dev

显示两个分支：

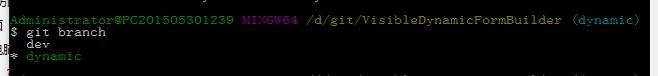


图 ‑7显示分支

可以利用git merge <branchname>的命令将其他分支合并到当前分支！

由此，分支建立成功；

### 3.2.2项目管理标准

遵循B.W.Boehm提出总结出了的七条原则

（1）用分阶段的生命周期计划严格管理；

（2）坚持进行阶段评审；

（3）实行严格的产品控制；

（4）采用现代程序设计技术；

（5）结果应能够清楚地审查；

（6）开发小组地人员应该少而精；

（7）承认不断改进软件工程实践的必要性。

项目管理相关文档标准使用GB8567-2006计算机软件文档编制规范，保证项目管理相关文档的规范性；

项目采用MVC架构（Model，View，Controller）。

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。

通常模型对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分。

通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分。

通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

### 3.2.3定期例会

为保证项目的风险评估、成本预算、客户沟通、需求分析、开发管理、产品质量、程序设计、产品交互正常有序进行，需要定期开展例会。

例会根据项目管理流程进行例会，是既科学又高效的例会模式。在软件项目管理流程的每一个流程中都开展中期例会和总结例会；中期例会主要完成系统概要性任务，总结任务即对该流程中完成的任务进行总结评估讨论；对于客户沟通任务和需求分析需要迭代进行的项目管理流程，需要按特殊要求制定更多例会。

本项目每个月进行一次系统性例会；

客户沟通过程例会，按每两个月与客户进行一次整体性沟通，提取用户需求和要求；

在软件生命周期中，也按周期定期举行例会，对于设计流程，可以缩短一半的间隔时间（即在一个月内开展两次例会）。

### 3.2.4Git使用规范说明

团队开发中Git的文件的命名规范使用是至关重要的。

1）本地Commit到服务器时，上传文件按类型选择相应分支；对于上传文件的命名，应该统一格式；例如：上传一个配置文件configure1.txt时，需要按格式

<Configure1.txt Date Member>

2）Commit文件后需要相应注释，注释顺序为：

<时间—Commit成员—文件类型—更改次数—细节注释>

### 3.2.5版本命名规则

正规的软件产品少不了良好的版本命名规规范。

可视化动态表单有四分部分组成：

1. 主版本号

2. 次版本号

3. 修订版

4. 日期版本号/希腊字母版本号

例如，本项目的版本之一：

V1.1.2.20171021\_Alpha

代表第一主版本，第一次版本，第二修订版本的2017年10月21日的内部测试版本；

### 3.2.6开发流程

（1）总体流程

计划阶段－>需求分析阶段－>总体设计阶段－>详细设计阶段—>软件开发阶段－>测试阶段－>部署产品与上线

（2）计划阶段

这个阶段主要工作有对项目进行可行性分析，制定项目计划草案和风险管理计划作，确定、分析项目风险并确定其优先级，还要制定风险解决方案。当确定开发之后则制定软件开发计划、人员组织结构定义及配备、过程控制计划。这个阶段主要编写以下文档：

a）项目可行性分析报告

全面深入地进行市场分析、预测。，研究确定市场，主要是产品竞争对手和自身竞争力的优势、劣势，以及产品的营销策略，并研究确定主要市场风险和风险程度。对资源开发项目要深入研究确定资源的可利用量，资源的自然品质，资源的赋存条件和开发利用价值。

b）项目计划草拟

项目计划草案应包括产品简介、产品目标及功能说明、开发所需的资源、开发时间和里程碑。

c）风险管理计划

就是把有可能出错或现在还不能确定的东西列出来，并制定出相应的解决方案。风险发现得越早对项目越有利。

d）软件开发计划

将开发过程中有关各项工作的负责人员、开发进度、经费预算、软硬件等问题做出的安排记载下来，以便根据本计划开展和检查本项目的开发工作。使项目成员和项目干系人了解项目开发计划书的作用、希望达到的效果。开发计划书的作用一般都是项目成员以及项目干系人之间的共识与约定，项目生命周期所有活动的行动基础，以便项目团队根据本计划书开展和检查项目工作。

e）配置管理计划

常见的人员组织结构有垂直方案、水平方案、混合方案。垂直方案中每个成员充当多重角色。水平方案中每个成员充当一到两个角色。混合方案则包括了经验丰富的人员与新手相互融合。具体选择根据人员实际技能情况进行选择。

（3）需求分析阶段

结合可视化动态表单的用户群体的特性，提取需求。使系统工作方面与用户达成一致。描述软件的功能和性能，确定软件设计的约束和软件同其他系统元素的接口细节，定义软件的其他有效性需求，借助于当前系统的逻辑模型导出目标系统逻辑模型，解决目标系统“做什么”的问题。这个阶段需要编写主要文档：

a）需求规格说明书

说明编写这份软件需求说明书的目的，指出预期的读者。软件需求说明书的作用在于便于用户、开发人员进行理解和交流，反映出用户问题的结构，可以作为软件开发工作的基础和依据，并作为确认测试和验收的依据。说明书的内容主要有客户需求的功能点、架构、运营模式。为软件系统概要及详细设计、软件编码、软件测试、测试用例的编制、质量和进度控制、验收提供依据。

需求规格说明书严格按照GB8567-2006计算机软件文档要求。

（4）总体设计阶段

结合需求分析，提取功能需求与非功能需求特性；结合用户的特殊性。给出系统的总体设计。并编写项目概要说明。

编写项目概要设计说明书，概要设计说明书主要用于对项目主要模块进行划分，以及各主要模块的偶合框架设计，系统数据流设计，系统用例说明及功能流程，运行环境及支持框架，包括运行平台选型、数据库选型、开发、测试工具选型、项目资源分配。

（5）详细设计阶段

根据总体设计，将功能分块，对每个模块进行详细设计。

模块包括登录模块、注册模块、管理修改系统/管理员模块、使用模块。

每个模块编写对应模块的详细设计分析说明书。详细设计说明书主要描述该模块内容，该模块内部各子模块的接口描述，模块对外耦合接口描述，模块内部的类图、数据流图、关系图、用例图及程序流程图设计，设计说明书视模块规模不同征对不同的主模块或者子模块。

（6）软件开发阶段

采用了面向对象方法，从物理上实现目标系统，完成项目。并且同时书写软件项目测试计划书。

a）软件架构

说明软件的组织结构、部署结构及运行环境。

b）功能设计

根据上一阶段的详细设计说明书、需求规约说明书等，依据提炼出相应的功能点，定义功能点之间的关联，完成软件原型设计。

c）数据库设计

根据上一阶段的数据库设计说明书，其中定义了项目中各个数据库表以及各个数据库之间的属性等，来完成定义数据库表之间的关联和各个表的字段的建立。

d）单元编码

根据之前所完成的文档，按照设计文档对各个单元进行编码。

e）集成系统

按软件组织结构的要求将各个子模块组合起来。

（7）测试阶段

软件测试是一种实际输出与预期输出间的审核或者比较过程。软件测试的经典定义是：在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其是否能满足设计要求进行评估的过程。

a）测试计划

根据之前完成的测试计划说明书，继续完成测试计划。收集和组织测试信息，为测试工作提供指导。

b）测试数据

根据测试计划中的，详细计划来使用测试数据进行测试，尽量使用真实数据。

c）测试报告

通过测试，记录测试结果，详细描述问题，提出解决办法并且记录结果。

（8）部署上线

部署产品，投入运行；至少经过5次的版本更正和修改后再进行部署，保证产品的可用性和效率。

### 3.2.7编码命名规则

（1）注释规则

所有源文件开头在靠头都有一个C语言风格的注释，其中列出类名、版本信息、日期和版权声明。如图 三-8；

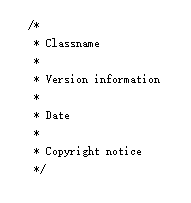


图 ‑8开头注释样例

（2）包和引入语句

Java源文件中，第一个非注释行是包语句，之后是引入语句。命名规则如图 三-9。

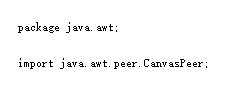


图 ‑9包及引入语句

（3）类和接口声明

类、接口声明对整个代码的可读性有很大影响。

1. 类/接口文档注释，使用（/\*.......\*/）
2. 类/接口实现注释，使用（/\*.......\*/）
3. 类的同属性变量间应该适当换行；类的不同属性变量必须换行；

（4）缩进排版

4个空格常被作为缩进排版的一个单位。缩进的确切解释并未详细指定(空格 vs. 制表符)。一个制表符等于8个空格(而非4个)。

（5）行长度

尽量避免一行的长度超过80个字符，因为很多终端和工具不能很好处理之。

注意：用于文档中的例子应该使用更短的行长，长度一般不超过70个字符。

（6）换行

当一个表达式无法容纳在一行内时，可以依据如下一般规则断开之：

- 在一个逗号后面断开

- 在一个操作符前面断开

- 宁可选择较高级别(higher-level)的断开，而非较低级别(lower-level)的断开

- 新的一行应该与上一行同一级别表达式的开头处对齐

- 如果以上规则导致你的代码混乱或者使你的代码都堆挤在右边，那就代之以缩进8个空格。

（7）注释

单行注释统一采用/\*.......\*/方式；

多行注释采用

/\*

......

......

......

\*/

行末注释统一采用//；

（8）布局

只在代码块的开始处声明变量。（一个块是指任何被包含在大括号"{"和"}"中间的代码。）不要在首次用到该变量时才声明之。这会把注意力不集中的程序员搞糊涂，同时会妨碍代码在该作用域内的可移植性。

（9）惯例

提供对实例以及类变量的访问控制。

若没有足够理由，不要把实例或类变量声明为公有。通常，实例变量无需显式的设置(set)和获取(gotten)，通常这作为方法调用的边缘效应 (side effect)而产生。

一个具有公有实例变量的恰当例子，是类仅作为数据结构，没有行为。亦即，若你要使用一个结构(struct)而非一个类(如果java支持结构的话)，那么把类的实例变量声明为公有是合适的。

（10）命名规范图示

|  |  |  |
| --- | --- | --- |
| 标识符类型 | 命名规则 | 例子 |
| 包(Packages) | 一个唯一包名的前缀总是全部小写的ASCII字母并且是一个顶级域名，通常是com，edu，gov，mil，net，org，或1981年ISO 3166标准所指定的标识国家的英文双字符代码。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门(department)，项目(project)，机器(machine)，或注册名(login names)。 | com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese |
| 类(Classes) | 命名规则：类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像URL，HTML) | class Raster; class ImageSprite; |
| 接口(Interfaces) | 命名规则：大小写规则与类名相似 | interface RasterDelegate; interface Storing; |
| 方法(Methods) | 方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。 | run(); runFast(); getBackground(); |
| 变量(Variables) | 除了变量名外，所有实例，包括类，类常量，均采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。 变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为i，j，k，m和n，它们一般用于整型；c，d，e，它们一般用于字符型。 | char c; int i; float myWidth; |
| 实例变量(Instance Variables) | 大小写规则和变量名相似，除了前面需要一个下划线 | int \_employeeId; String \_name; Customer \_customer; |
| 常量(Constants) | 类常量和ANSI常量的声明，应该全部大写，单词间用下划线隔开。(尽量避免ANSI常量，容易引起错误) | static final int MIN\_WIDTH = 4; static final int MAX\_WIDTH = 999; static final int GET\_THE\_CPU = 1; |

# 项目管理中配置管理工具的应用

## 4.1配置管理工具GIT的安装与初始化

各个组员均使用Windows操作系统进行系统实现，从Git官网上直接下载安装程序进行安装。

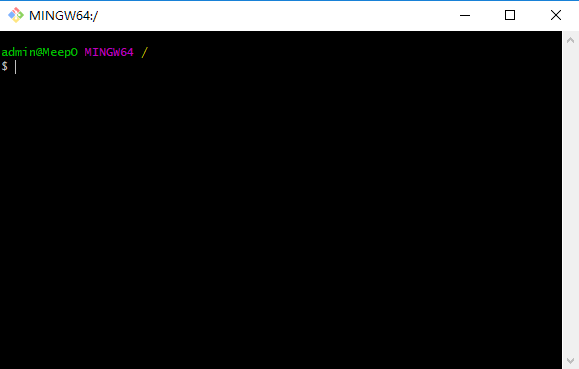


图 ‑1 下载并安装Git（打开Git bash）

在合适的文件夹中使用

git init

指令

初始化git

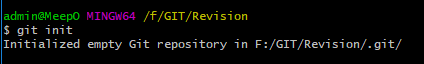


图 ‑2 初始化Git仓库

配置所使用机器的用户名和用户邮箱



图 ‑3 配置所使用机器的用户名（杨一帆）



图 ‑4 配置所使用机器的用户邮箱（杨一帆）

其他组员的用户名和用户邮箱为：

杨一帆：IvanMee ivanyang.yyf@foxmail.com

乔世超：MichaelJoe michael\_joe1023@outlook.com

别明洲：DianaLuna [1103593045@qq.com](mailto:1103593045@qq.com)

王子昊：

郭嘉威：

张瑶：

## 4.2创建版本库

通过git init命令把这个目录变成Git可以管理的仓库

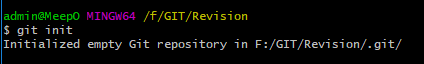


图 ‑5 将所在的Revision目录变成可管理的仓库



图 ‑6 当前目录下生成了一个.git的目录（用于追踪管理版本库）

测试版本库：

使用note++编写测试文件，保存为gittest.txt保存在Revision目录中

GITTEST

GITTEST

GITTEST

使用命令 git add 将文件添加到仓库：

git add gittest.txt



图 ‑7 将测试文件添加到仓库中

使用命令git commit 将文件提交 //引号中为版本提交说明

git commit -m “test 1.0”

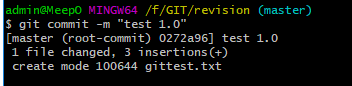


图 ‑8 将文件提交到仓库

测试文件修改和版本回溯：

修改gettest.txt

GITTEST

GITTEST

GITTEST

GITTEST

GITTEST

再次提交

git add gettest.txt

git commit -m “add 2lines”

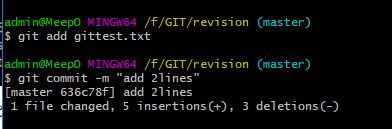


图 ‑9 尝试文件修改和再次提交

查看版本历史纪录

git log

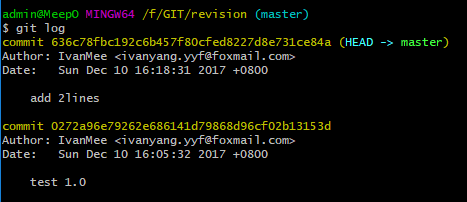


图 ‑10 查看版本历史记录

回溯到上一个版本：

git reset --hard HEAD^



图 ‑11 回溯到上一个版本

查看gittest.txt

cat gittest.txt

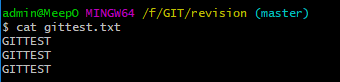


图 ‑12 输出上一个版本

删除文件及版本库中文件

git rm gittest.txt

git commit -m “remove gittest.txt”

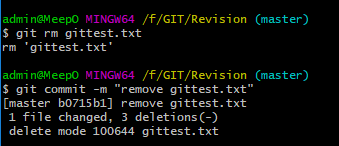


图 ‑13 删除文件及版本库中文件

## 4.3建立远程仓库

在这次的系统配置管理之中，我们选择了GitHub网站作为远程仓库，GitHub提供了Git仓库托管服务，我们在注册了之后就可以免费获得Git远程仓库。

由于本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以我们需要现在自己的电脑上进行配置。

4.3.1创建SSH KEY

使用ssh-keygen -t rsa -C

ssh-keygen -t rsa -C “ivanyang.yyf@foxmail.com”

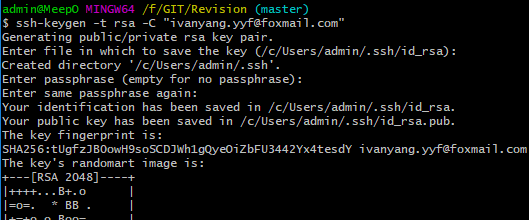


图 ‑14 创建SSH KEY（未设置密码）

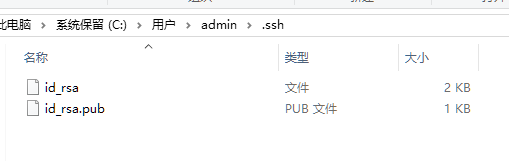


图 ‑15 在用户目录里内衣找到.shh目录

其中id\_rsa是私钥不能泄露出去

id\_rsa.pub是公钥可以分享

4.3.2登录GitHub 设置自己的ssh key

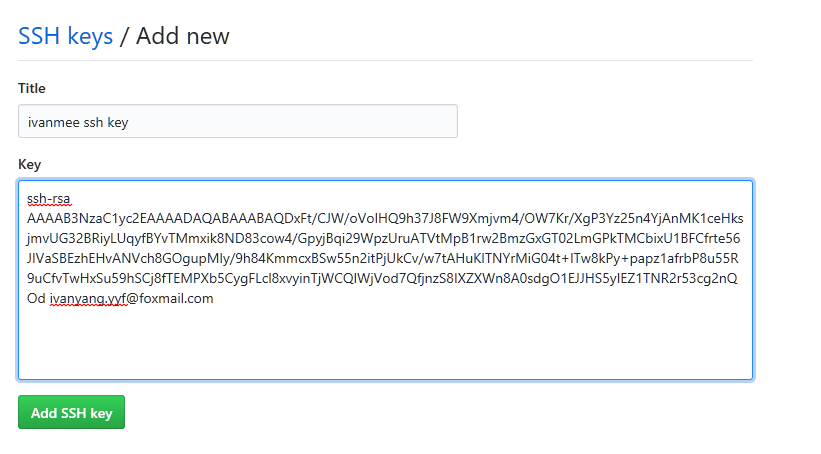


图 ‑16 添加自己的ssh key

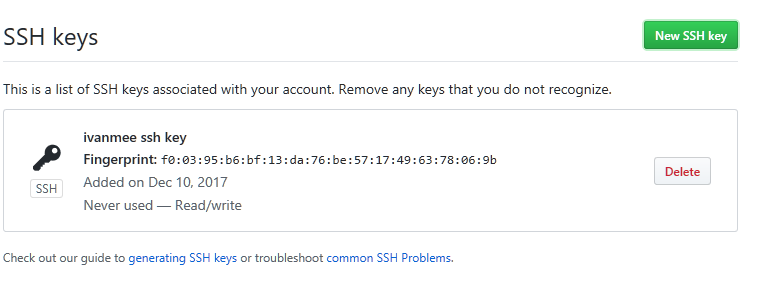


图 ‑17 添加成功后可以查看到自己的ssh key

添加远程库

由一名成员在GitHub中创建Git仓库

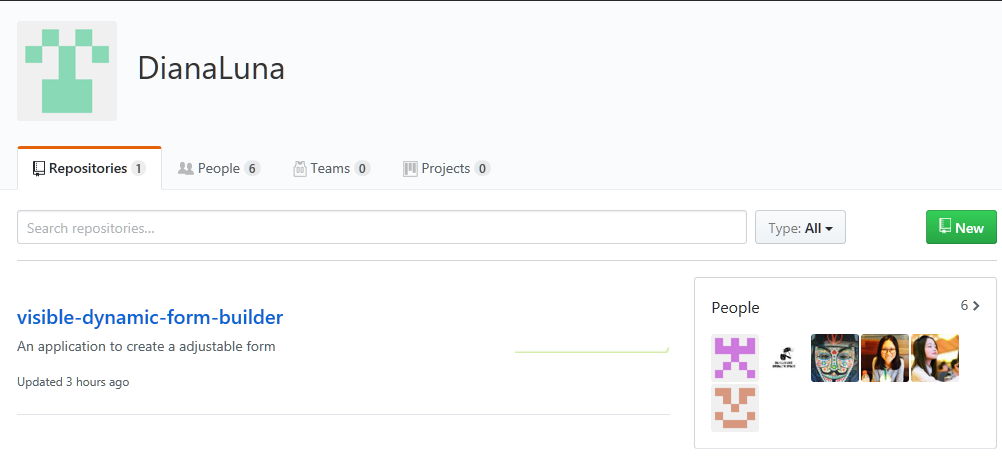


图 ‑18 创建Git仓库（小组）

将已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库



图 ‑19 将本地仓库与GitHub仓库关联

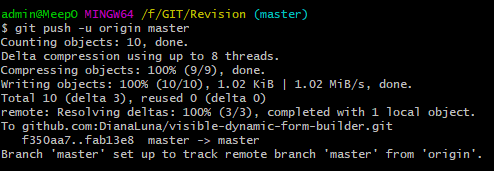


图 ‑20 将本地库的内容推送到远程库上

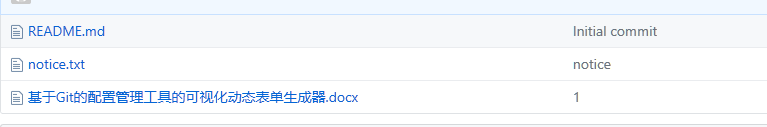


图 ‑21 成功推送notice.txt

同理，推送项目。

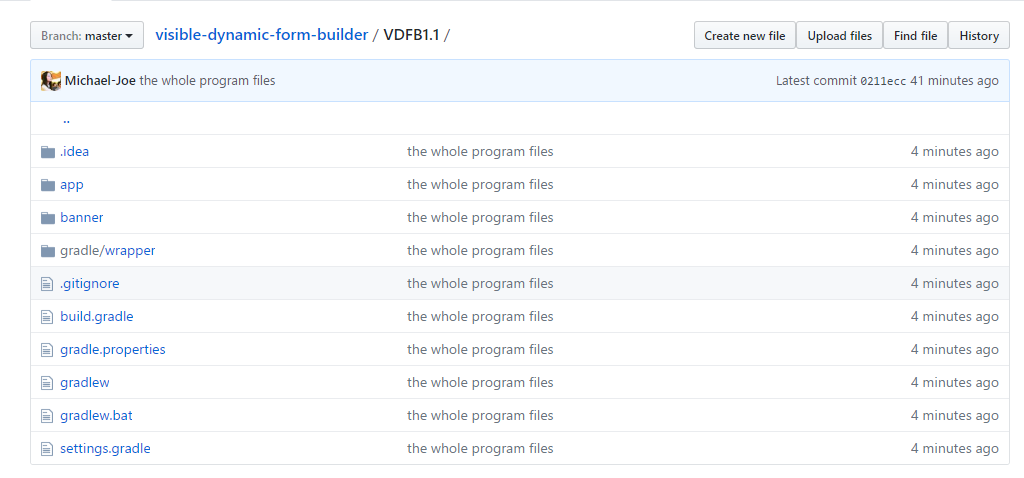


图 四‑22

小结：

在每次的本地开发完成后，我们将代码信息保存至本地仓库，用git命令

git add

git commit -m “”

将文件提交到仓库，并用git命令

git push origin master

将本地master分支的最新修改推送给GitHub的版本库中。

我们现在已经可以使用分布式版本库进行高效地团队编程了。

# 开发质量控制

需求分析要把软件计划期间建立的软件可行性分析求精和细化，分析各种可能的解法，并且分配给各个软件元素。需求分析是软件定义阶段中的最后一步，是确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求。

软件需求分析所要做的工作是深入描述软件的功能和性能，确定软件设计的限制和软件同其它系统元素的接口细节，定义软件的其它有效性需求。进行需求分析时，应注意一切信息与需求都是站在用户的角度上。尽量避免分析员的主观想象，并尽量将分析进度提交给用户。在不进行直接指导的前提下，让用户进行检查与评价。从而达到需求分析的准确性。

分析员通过需求分析，逐步细化对软件的要求，描述软件要处理的数据域，并给软件开发提供一种可转化为数据设计、结构设计和过程设计的数据和功能表示。在软件完成后，制定的软件规格说明还要为评价软件质量提供依据。

我们小组在进行需求分析的时候，我们通过用户需求和功能需求（也包括非功能需求）的角度来进行具体的分析。

在进行用户需求的时候，我们使用了用例图等UML手段来进行描述我们可视化动态表单的分析。在说明书中，我们描述了系统该有的外部行为。作为功能需求的补充，我们还进行了非功能需求的讨论，我们描述了系统展现给用户的行为和执行的操作，还有外部界面的具体细节，性能要求和需要实现的约束条件及质量属性。

在需求分析阶段我们也要进行设计用户验收测试，这样会让我们在之后的维护阶段减少所需投入的精力。还有就是用反向思维，来从后往前的进行项目的需求分析。

## 5.2设计阶段

我们小组在设计阶段就进行了详细的角色分工。我们设计对需求进行了较高的覆盖率。每个小组成员对自己的模块进行了详细的划分，每位同学都约定了今后交互模式，保证了我们快速迭代的开发进度。负责测试的同学也开始讨论了系统的可测试性以及自动化测试的工具编写。并且开始试着设计系统测试，负责维护的同学考虑了系统的扩展性，为了满足不同角色的不同需求，我们进行软件设计的时候，我们进行了全面的考虑：

1. 功能性：

包括完全性、正确性、安全性、兼容性、互用性。完全性包括功能点覆盖率，重点功能点覆盖率，优先功能覆盖率。正确性包括需求一致度。安全性根据软件需求的不同有不同的安全性要求。

1. 效率：

包括产品运行的时间效率和利用的硬件资源也就是时间特性、资源特性两方面来考虑。

1. 维护性：

包括架构的可改正性，可扩充性以及可测试性。如果用户的一个很小的需求变更会引起架构设计很大的变化，那么这样的架构设计的可改正性和可扩充性就比较差。

1. 可移植性：

包括硬件的独立性、软件独立性、可安装性、可重用性。软件设计是否模块化、每个模块的可复用性如何都是应该考虑的因素。

1. 可靠性：

包括缺陷数量、容错性、可用性。

1. 使用性：

包括可理解性、易学习性、可操作性、易沟通性。我们软件的最终目的是让用户来使用的，如果易用性不好，可操作性不好都会影响用户对软件的接纳程度。因此在软件的可使用性也是非常重要的。

## 5.3编码阶段

我们小组在编码阶段，最看重的标准就是代码的可读性以及规范性。这里的可读性并不是简单的代码，而是容易理解的代码，因为过于复杂的代码会导致我们在测试和维护阶段难以展开相应的工作，同时出错的概率也会变得很高。我们小组规定我们每个方法的内部代码不会太长，并且不使用太多难以理解的数据集，以此来避免维护阶段的困难。只有让同学能很有效的分析它们，才不容易出现缺陷和错误。

还有就是如果类与类之间有太高的耦合度，一旦其中某个类发生改变的时候，其它的类就会发生意向不到的变化。在这点上，我们从导入类的个数来判断类之间的耦合度，如果导入类的个数很多，就不免会出现其中某个类影响这个类本身。最后就是避免该类的public方法太多。

虽然写出可读的、规范的代码会增加同学的工作负担。的确，短时间来说，给自己的代码加上注释是要花些额外的时间。但是，在小组协作中，注释过的代码，会方便我们小组每个成员的每个阶段进行的更加顺利。从项目的长远角度来看，这是非常值得我们去做的。

最后我们也希望小组成员能够从提升效率的角度出发，来进行相应代码的编写。比如尽量保证代码模板化，这样不经增加规范性，而且可以使小组成员在代码交流方面变得非常顺畅。然后我们也希望小组成员能够把功能放到相同的位置，再配上相应的文档说明，这样会使成员在编写的时候不用再写一套方法。这样不仅能提升效率，还能避免每个成员在同一个方法上因为编码的不同而导致不必要的错误。

## 5.4测试阶段

重点关注单元测试，在开发过程中越早的发现缺陷，修复它就越容易、越快速、越便宜。对藏身于代码中的缺陷来说，单元测试是发现它们最早的阶段。在进行单元测试的时候，我们需要对软件中的最小可验证单元进行检查和验证，就像类和方法的测试。

进行单元测试之后，我们才利于重构、简化集成、容易编写文档，并且及早用于设计。虽然不能覆盖所有的执行路径，但是我们会在后续的测试中，慢慢发现新的错误。关于单元测试的代码编写，我们安排固定的成员，这样会使效率有所提升。

因为我们每天保证必要的交互，并且在之前的阶段我们也注意着后期的测试。所以在测试阶段我们进行要比想象中顺利。

我们也进行了集成测试，将所有的模块组装起来后，主要检查接口是否存在问题，组装后的整体功能和性能表现。然后利用黑盒和白盒测试方法，检查源代码是否符合开发规范，接口是否存在问题，整体功能有无错误、界面是否符合设计规范、性能是否满足用户需求等。  
5.5项目实施阶段

我们会对系统进行及时跟进。如果需要进行需求的变更，我们会及时进行更改。尽早的将项目实施也会让我们尽早看到组员们对项目的理解到底存在什么样的偏差。

在实施之后我们会安排同学进行使用，来收集反馈，如若出现任何重大偏差，我们会立马进行相应的更改。即使没有检查出严重错误，我们我们也会做好文档的记录。

## 5.6软件质量保证SQA中的7个方面的要求详细解释

### （1）定义恰当的质量目标

软件最终是要交付给用户使用的，因此应从用户的角度来定义软件质量目标，软件应满足用户的业务需求，实现令人满意的用户体验。

好处：既不将质量目标定得太高，任由你付出百般努力也无法实现，也不将目标定的过低，那样你无法给用户交差，根据时间，资源和预算客观情况定义合适的软件质量标准最好，既不让开发团队感觉痛苦，又能让用户满意。

### （2）让每个人都知道质量的重要性

尽量在软件开发生命周期的前段时间减少软件缺陷，避免在后期来消灭缺陷，那样耗费的时间和精力更多。

好处：让每个人都知道质量的重要性后，他们就会从心理上更注重代码质量，就会更用心写出高质量的软件。

### （3）调整团队个人的目标，纳入质量考核体系

根据业务需求调整团队和个人的工作目标，并纳入质量考核体系，实施严格的奖惩措施，刺激开发人员的工作效率和工作质量。

好处：根据团队成员的执行表现给予适当奖励，让他们知道改善软件质量是一种奋斗目标，逐渐发展成为一种习惯。

### （4）获取正确的需求

确保从需求获取开始，项目就朝正确的方向迈进，需求偏离或需求错误是让开发人员最头痛的事，大量的返工和修改会熄灭本已燃起的激情，而正确的需求会给开发人员带来愉快的心情。

好处：减少返工和重新测试周期，减少总体工作量。

### （5）将测试重点放在最关键和风险很高的点

在时间有限的情况下，不可能将方方面面的缺陷通过测试全部暴露出来，这时只有抓住重点，做到有的放矢，将核心功能点重点测试，避免重大缺陷成为漏网之鱼。

好处：杜绝关键缺陷，即便有其它缺陷未被发现，也不至于影响到软件的整体质量。

### （6）提高设计质量

开发人员会根据架构师的设计文档进行编码的，如果设计描述得含混不清，那开发人员可能会根据自己的理解编写代码，或许就会造成南辕北辙的结果。

好处：参照简明清晰的设计编写出来的代码也会更简单，更干净，也更容易测试和返工，代码中包含的错误也会更少，也更容易诊断和修复缺陷。

### （7）合理使用自动化测试工具

传统的手工测试很难覆盖软件的全部功能点，某些后台功能只能借助工具来测试，此外，手工测试的效率低，反复单调的测试更是对测试人员心理素质的极大考验，容易造成对测试工作的懈怠，降低测试质量。

好处：通过自动化测试工具的合理使用，可以缩短测试周期，提高测试的可重复性。

# 开发效率保证

## 6.1保证开发效率的意义

开发效率通常反映在两点上：

1.开发新功能是否迅速；

2.修复缺陷是否及时。

由于我们这次的产品将由产品快速迭代增量开发的敏捷团队来完成，项目规模也并不大，相对于同类型项目较低的复杂度决定了这样的产品应当适量辅以一些局部地深入挖掘以及创新。所以这样的系统需要高速的开发效率，在发现用户需求的时候快速增量开发新功能并且及时修补漏洞。

小型的动态表单设计器并不具有复杂度的优势。这种时候，相对较高的开发效率和迅速灵活的反应能力就成了生死存亡的关键。

在这种情况下，对于实现人员而言，最重要的素质在于审慎地控制复杂度，绝大部分情况下不要主动选择繁复的实现方式，避免在无谓的细节上纠缠不清，尽量采用业界惯常做法(必要时可适当简化)。 通过设计和流程上的管控 (而非细枝末节的优化)，让运行时的性能始终保持在可控的安全区域以内，是省心省力且比较有效的手段。

## 6.2软件开发中的心态

第一种：舍得心态。

前期做了大量细致的工作，在发现所做工作达不到客户需求或与团队目标有出入的情况下。虽然自己认为也可以，但也能够果断舍得并丢弃，从头再来。

第二种：团队合作心态。

遵守团队规划，保证个人的文档或代码编写不自成一家，结构或风格与团队总体高度一致。避免犯“自以为是，我行我素”等低级错误。

第三种：目标专一心态。

在工作中，紧抓目标不放，不会因为太注重局部细节，而忘了整体目标；在工作中，为了解决某一问题而进行资料查找时偶然发现了一个自己很感兴趣的话题或自己认为 很重要（也可能将来很有用）的话题，并立刻追随这一话题进行大量的工作，而忘了最 初的原始目标。优秀的软件开发人员可以抵得住这种诱惑，紧紧抓目住标不放，从而使 自己一直保持在高效工作状态。

第四：大是大非毫不含糊的心态。

在工作中，知错就改，乐于接受不同意见。不在明知自己有不妥，或团队成员有更好的解决方案的情况下，因自尊心太脆弱，不愿直面，仍固执己见不愿妥协。

第五：客户第一的心态。

只要是用户（市场）的合理需求，想方设法去达成它，追求完善，不怕重复劳动。

## 6.3提高开发效率

提升开发效率是每个团队都要面对的问题。所有提升效率的办法无非就是开源节流。对于我们这个快速迭代敏捷开发团队来说，我们需要的是及时的内部的沟通。但是，这不代表着我们需要频繁的开会。

我们的每次沟通，都有明确的目标，我们要及时反馈每个人问题，并形成成员交互。然后我们要及时确定方向，避免成员们分散自己的开发精力，或者对开发方向产生不同的理解，导致用力方向不同导致我们没有办法能够集中起来进行单个问题的攻克。

在确定开发方向之后，我们尽可能提升自己的绝对开发时间。避免每天无效的开发时间。当然团队协作肯定需要我们的每个人都要及时呈递自己的问题。这样我们才能去统筹分析如何去攻克一个一个困难。还有就是在频繁交流之后，会逐渐降低我们的学习成本，这样我们以在测试环节减少大量的重复学习时间。

从框架的角度出发，我们提升效率可以从以下角度入手：

（1）代码模板化

框架一般都有统一的代码风格，同一分层的不同类代码，都是大同小异的模板化结构，方便使用模板工具统一生成，减少大量重复代码的编写。在学习时通常只要理解某一层有代表性的一个类，就等于了解了同一层的其他大部分类结构和功能，容易上手。小组中不同的同学采用类同的调用风格进行编码，很大程度提高了代码的可读性，方便维护与管理。

（2）重用

开发框架一般层次清晰，小组成员开发时都会根据具体功能放到相同的位置，加上配合相应的开发文档，代码重用会非常高，想要调用什么功能直接进对应的位置去查找相关函数，而不是每个组员各自编写一套相同的方法。

（3）高内聚（封装）

框架中的功能会实现高内聚，开发人员将各种需要的功能封装在不同的层中，给大家调用，而大家在调用时不需要清楚这些方法里面是如果实现的，只需要关注输出的结果是否是自己想要的就可以了。

（4）规范

框架开发时，必须根据严格执行代码开发规范要求，做好命名、注释、架构分层、编码、文档编写等规范要求。因为你开发出来的框架并不一定只有你自己在用，要让别人更加容易理解与掌握，这些内容是非常重要的。

（5）可维护

成熟的框架，对于二次开发或现有功能的维护来说，操作上应该都是非常方便的。比如项目要添加、修改或删除一个字段或相关功能，只需要简单的操作，十来分钟或不用花太多的工夫就可以搞定。新增一个数据表和对应的功能，也可以快速的完成。功能的变动修改，不会对系统产生不利的影响。代码不存在硬编码等等，保证软件开发的生产效率和质量。

（6）协作开发

有了开发框架，我们组才能更好的进行协作开发，成熟的框架将大大减轻项目开发的难度，加快开发速度，降低开发费用，减轻维护难度。

我们小组秉持“小步快跑，快速迭代”的理念。虽然组员人手有限，但因为目标的集中和力量的集中，我们在开发的过程中，总可以一点一点的做出成果。这可以极大的鼓舞组员们的信心，并激励着大家再接再厉。持续每天发现问题，并且解决问题。这是我们每天坚持做的事情。在细微的切分之下，我们能够一步一步的完成项目的进度，在稳扎稳打之中保证了我们的开发效率。

对于每个个人来讲，要想最终提升小组的开发效率，我们每个人都需要提升自己的代码编写能力。保证自己所写的代码的正确性、高效性、健壮性、可移植性。从下而上的提升效率，最终保证项目进度的顺利进行。

为了保证成员们协作方便，我组在本次开发中使用了GitHub来作为远程仓库。这样保证我们每个组员的交互可以更加方便。

## 6.4降低维护开销

降低维护开销可以分担我们的开发压力。同时也提升我们的效率，让我们能把能够更快的进行下一步的开发过程。

良好的功能需求文档

只有当我们把需求文档化，我们才能够消除项目不能满足最终所需的风险。只要我们每天所做可以满足归档的需求，我们就不会导致项目返工、项目延期、增加成本以及软件团队产生矛盾的风险。这样我们在之后的维护中，我们才不用耗费太多精力。

尽早明确非功能的需求

为了避免我们的返工是因为在非功能的领域，我们需要一个清晰的、尽早定义的非功能的需求文档。只有这样，我们才能避免做出的项目，因为这些非功能的需求而导致难用、速度慢、不安全、不可靠、扩展性太差等诸多问题。

集中在缺陷预防而不是测试

项目开始时修正一个错误远比开发完成后再修正更快速更便宜。在项目越后期来发现一个错误，来修正他所需的花费就越多。所以我们降低维护开销的一个重要的手段就是在前期组员交流时仔细考虑项目的需求。这样我们才源头上降低维护成本。

使用工具来发现错误

在开发的早期阶段就来开发一套自动化的测试工具，这会让我们很快的发现潜在的缺陷，为了以后的测试阶段节省宝贵的时间。

早期对部分系统的性能测试可以暴露设计很差的模块，它发布后会拖累整个系统的性能到用户的容忍度以外。”这种情况下，该模块很容易被确认并被隔离，很容易修正或者重写。如果在系统交付测试后或者发布前才去做性能测试，这种确定过程将困难很多

# 总结与展望

在项目开发中，我们使用了Git，与SVN相比，Git分布式的特点更便于我们进行项目的开发管理。

Git操作命令包括：clone，pull，push，branch ，merge ，push,rebase，擅长的是程序代码的版本化管理。它 是一款免费的、开源的、分布式的版本控制系统。旨在快速高效地处理无论规模大小的任何软件工程。每一个 Git克隆 都是一个完整的文件库，含有全部历史记录和修订追踪能力，不依赖于网络连接或中心服务器。其最大特色就是“分支”及“合并”操作非常快速、简便。

从一般开发者的角度来看，Git有以下功能：从服务器上克隆完整的Git仓库（包括代码和版本信息）到单机上，在自己的机器上根据不同的开发目的，创建分支，修改代码、在单机上自己创建的分支上提交代码。在单机上合并分支。把服务器上最新版的代码fetch下来，然后跟自己的主分支合并。生成补丁（patch），把补丁发送给主开发者。看主开发者的反馈，如果主开发者发现两个一般开发者之间有冲突（他们之间可以合作解决的冲突），就会要求他们先解决冲突，然后再由其中一个人提交。如果主开发者可以自己解决，或者没有冲突，就通过。一般开发者之间解决冲突的方法，开发者之间可以使用pull 命令解决冲突，解决完冲突之后再向主开发者提交补丁。

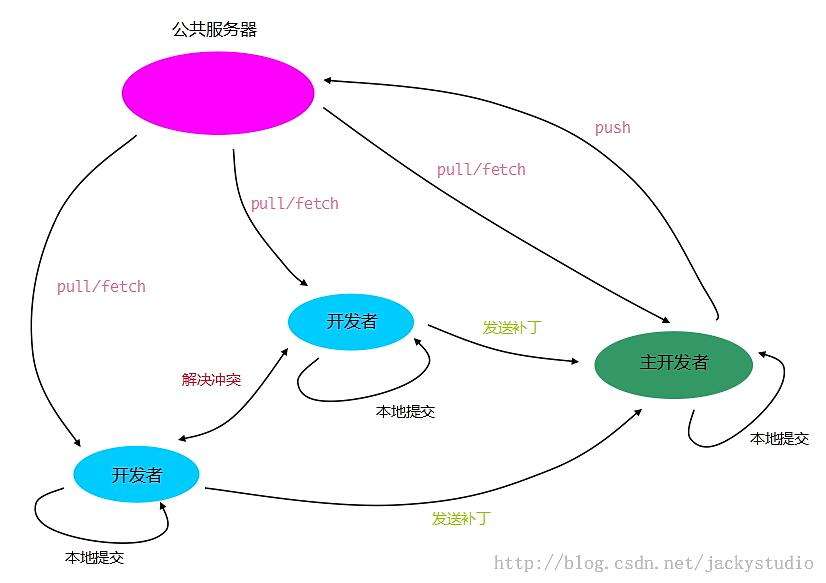


图 七‑1

Git工作过程

Git具有以下特点：Git中每个克隆(clone)的版本库都是平等的。你可以从任何一个版本库的克隆来创建属于你自己的版本库，同时你的版本库也可以作为源提供给他人，只要你愿意。Git的每一次提取操作，实际上都是一次对代码仓库的完整备份。提交完全在本地完成，无须别人给你授权，你的版本库你作主，并且提交总是会成功。甚至基于旧版本的改动也可以成功提交，提交会基于旧的版本创建一个新的分支。Git的提交不会被打断，直到你的工作完全满意了，PUSH给他人或者他人PULL你的版本库，合并会发生在PULL和PUSH过程中，不能自动解决的冲突会提示您手工完成。冲突解决不再像是SVN一样的提交竞赛，而是在需要的时候才进行合并和冲突解决

SVN是Subversion的简称，是一个开放源代码的版本控制系统,支持大多数常见的操作系统。作为一个开源的版本控制系统,Subversion管理着随时间改变的数据。这些数据放置在一个中央资料档案库(repository)中。这个档案库很像一个普通的文件服务器,不过它会记住每一次文件的变动。这样你就可以把档案恢复到旧的版本,或是浏览文件的变动历史。Subversion是一个通用的系统,可用来管理任何类型的文件,其中包括了程序源码。

[集中式管理](http://baike.baidu.com/view/3794518.htm)的工作流程如下图：

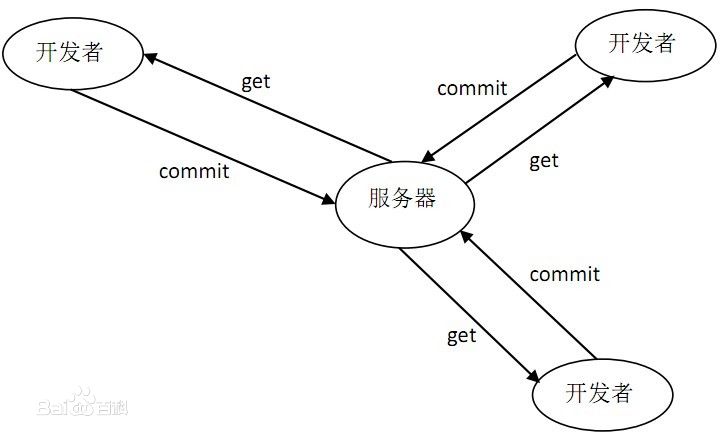


图 七‑2

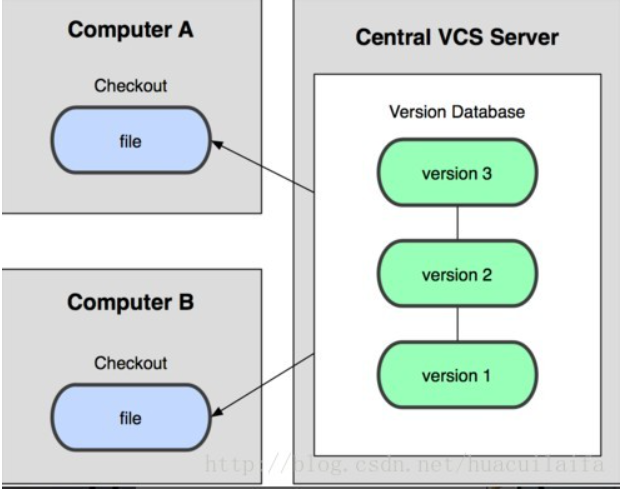


图 七‑3

集中式管理

集中式代码管理的核心是[服务器](http://baike.baidu.com/view/899.htm)，所有开发者在开始新一天的工作之前必须从服务器获取代码，然后开发，最后解决冲突，提交。所有的版本信息都放在服务器上。如果脱离了服务器，开发者基本上可以说是无法工作的。

Git是一个开源的分布式版本控制系统，用以有效、高速的处理从很小到非常大的项目版本管理。Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

分布式相比于集中式的最大区别在于开发者可以提交到本地，每个开发者通过克隆（git clone），在本地机器上拷贝一个完整的Git仓库。

下图是经典的Git开发过程：

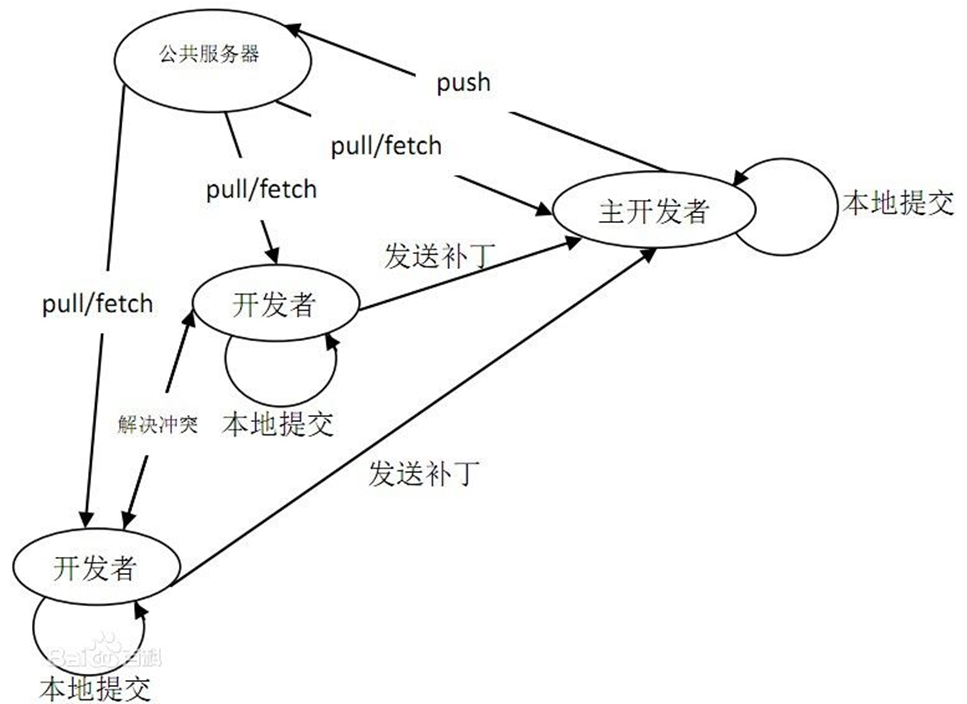


图 七‑4

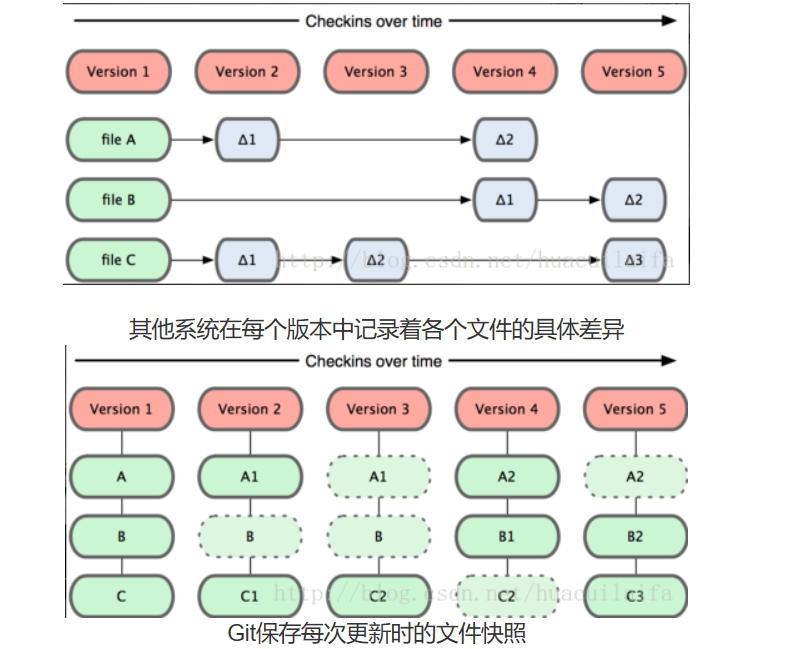


图 七‑5

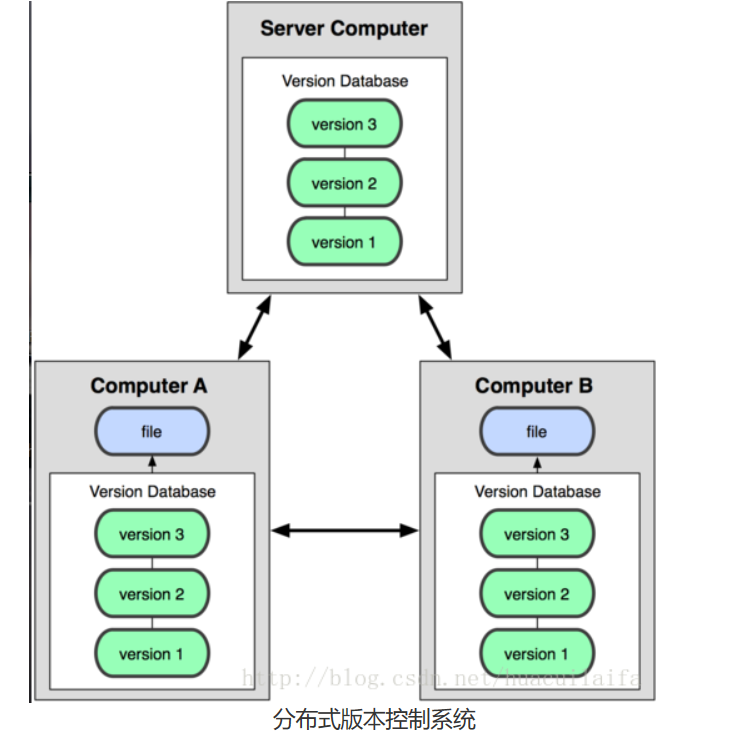


图 七‑6

与SVN相比，GIT的优点：

1.版本库本地化，支持离线提交，相对独立不影响协同开发。每个开发者都拥有自己的版本控制库，在自己的版本库上可以任意的执行提交代码、创建分支等行为。例如，开发者认为自己提交的代码有问题？没关系，因为版本库是自己的，回滚历史、反复提交、归并分支并不会影响到其他开发者。

2.更少的“仓库污染”。git对于每个工程只会产生一个.git目录，这个工程所有的版本控制信息都在这个目录中，不会像SVN那样在每个目录下都产生.svn目录。

3.把内容按元数据方式存储，完整克隆版本库。所有版本信息位于.git目录中，它是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签、分支、版本记录等。

4.支持快速切换分支方便合并，比较合并性能好。在同一目录下即可切换不同的分支，方便合并，且合并文件速度比SVN快。

5.分布式版本库，无单点故障，内容完整性好。内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

基于Git的以上功能和特点，并与SVN进行比较，我们选择以Git为工具进行软件项目配置管理。

软件配置管理是指通过执行版本控制、变更控制的规程，以及使用合适的配置管理软件，来保证所有配置项的完整性和可跟踪性。配置管理是对工作成果的一种有效保护 。软件配置管理对于软件开发管理是如此重要，它的主要思想和具体内容在于版本控制。版本控制是软件配置管理的核心思想之一，是指对软件开发过程中各种程序代码、配置文件及说明文档等文件变化的管理。做好软件配置管理还会给项目经理带来许多方便，有利于对整个开发团队进行管理，但管理的效果还要取决于整个开发团队成员的配合程度。 软件本身是思维逻辑和数据的固化，而思维逻辑和数据是无形的，软件配置管理就是对这些无形的思维逻辑和数据进行变更记录，使之可以度量、统计分析和管理。 只有做好了软件配置管理，我们才能避免在软件开发中使项目管理、风险管理、质量管理、过程管理成为空中楼阁，我们才能脚踏实地走向软件开发规范化管理的高峰。

在项目进行的过程中，我们不仅领会到了Git的优点，也明白了项目配置管理的重要性。同时，我们也认识到了团队协作在项目开发中的重要性。即使有完备的工具和扎实的编码能力，仅有一个人是无法高效完成大型项目的，良好的团队协作不仅能发挥每个人的优势，也可以开拓软件开发的思路，起到事半功倍的效果。