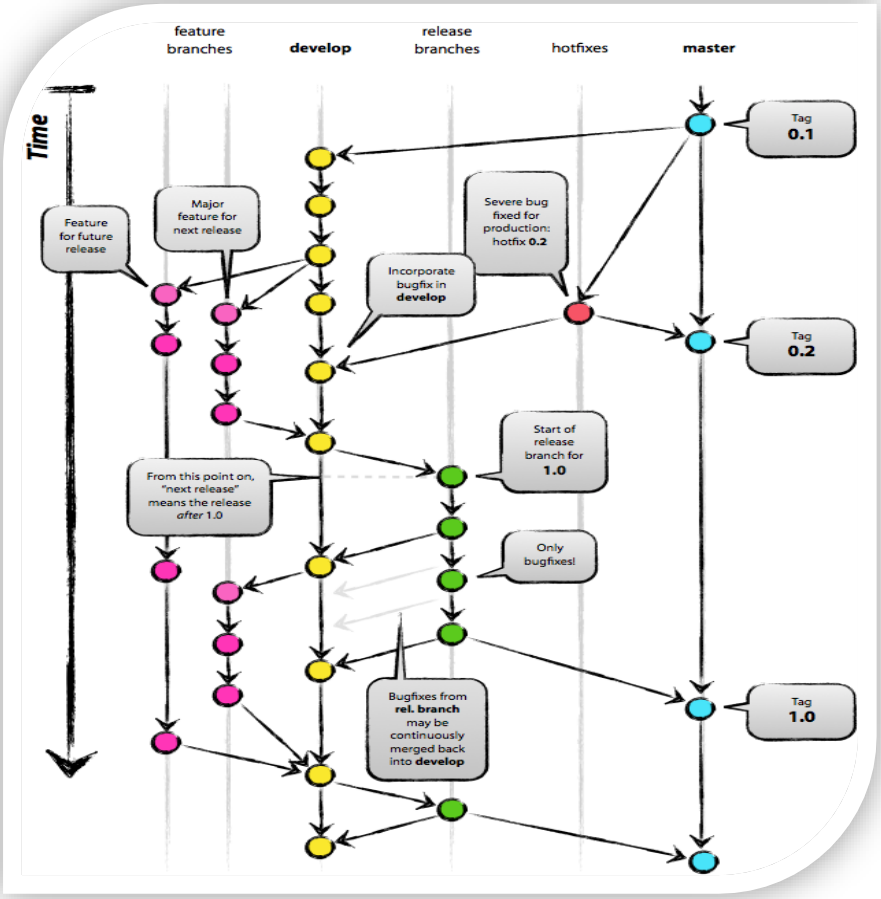
# 代码管控平台需求分析及现状评估报告




2018-12-10

## 前 言

本文基于分析公司的软件管控现状，结合代码管控理论基础 DevOps 的应用领域，对比 Microsoft Azure DevOps、IBM Jazz 和 GitLab 三大平台，进行技术选型，选择最适合于公司的软件管控协议和软件管控平台为 GitLab。最后根据公司的代码管控需求制订了项目任务分步实施计划。

# 代码管控平台需求分析及现状评估报告

（ABC 信息技术有限责任公司）

## 1　现状与需求

### 1.1公司现状

ABC 公司信息化建设发展已逐渐从基础设施建设向软件系统集成跨越发展，特别是近年来随着互联网技术日新月异，极大促进传统行业利用软件技术手段提高生产力。截止 2017 年，ABC 公司共建有信息系统 334 套，据不完全统计，ABC 自研发系统 148 套以上，占总系统量 44%以上，但其中经过代码成果质量严格评测并提交代码进行保存转化和再利用的不超过 10 套，占比 6.7%。

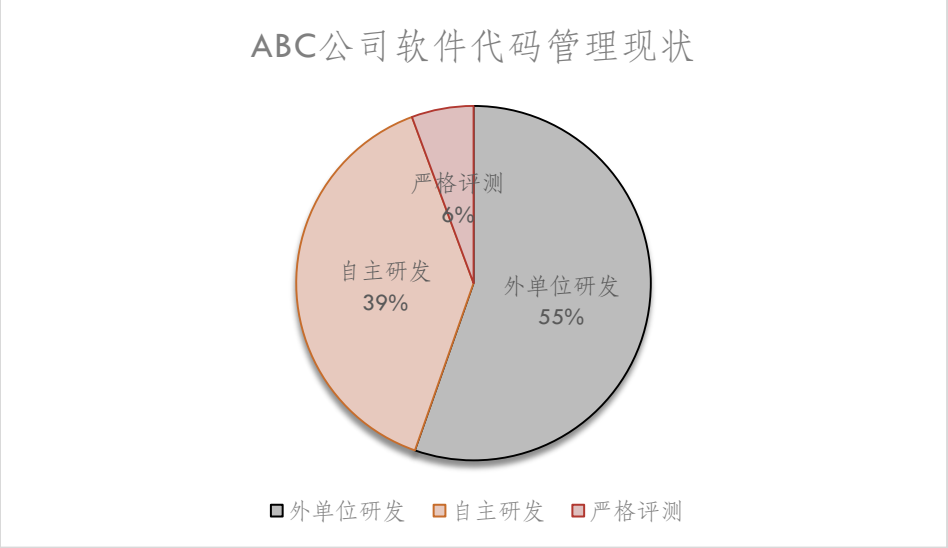



图 1.1 软件研发比重

软件成果属于知识型成果，具有一定的抽象性，他更多是以文件和程序的方式存在于计算机内。对于软件项目来讲，整个项目的投入最终的知识成果累积核心就在最后交付的成果代码和文档，此成果承载着公司的生产管理业务逻辑以及算法经验，是软件项目最重要的资产。对于最重要的成果之一，ABC 公司目前的管控手段是由开发方提交成果，人工归档，没有统一的标准对代码成果的真实性、有效性、品质并不做更多的检测评审。代码成果入库及出库的随机性导致软件知识成果大量流失，没有代码、低品质代码，代码作为个人或开发单位的私产被保存的情况大量存在。导致以下几种现象：

**第一：增大软件持续开发难度。**软件的存在是为了满足企业生产经营管理需要，随着生产技术和管理方法的改进，软件也要持续同步改造。我们不难想象没有代码、代码质量很低将对我们的业务带来什么影响？越往后需求响应周期越长，甚至无法响应。案例 1：某公司 2015 年的系统 2016 更换开发公司后全部重新开发。案例 2：仅 2014 年以来，有 5-6 个系统因为硬件环境改变导致系统无法部署继续运行（只有 1 个自开发系统修改代码后可正常运行）。

**第二：原系统追溯成本高，甚至高于重新开发成本。**因为原代码的缺失，若在生产过程中发生了数据泄密，或历史数据丢失，要想追溯往往需要 10 倍以上的成本。案例 1：某油田系统开发费 3 万元，取历史数据 400 万元。

**第三：知识产权流失，数据失泄密风险高。**2014 年信息安全评估结果，5 个系统因代码缺陷无法通过，被迫停止运行，重新改进。

## 1.2为什么需要进行软件代码管控？

**熵增定律**也叫热力学第二定律，**熵是指一个系统的无序度，或者是无效的负能量**。一个孤立系统的熵（即负能量）永远是增加的，并且将两个系统连接在一起时，其合并系统的熵大于所有单独系统熵的总和。

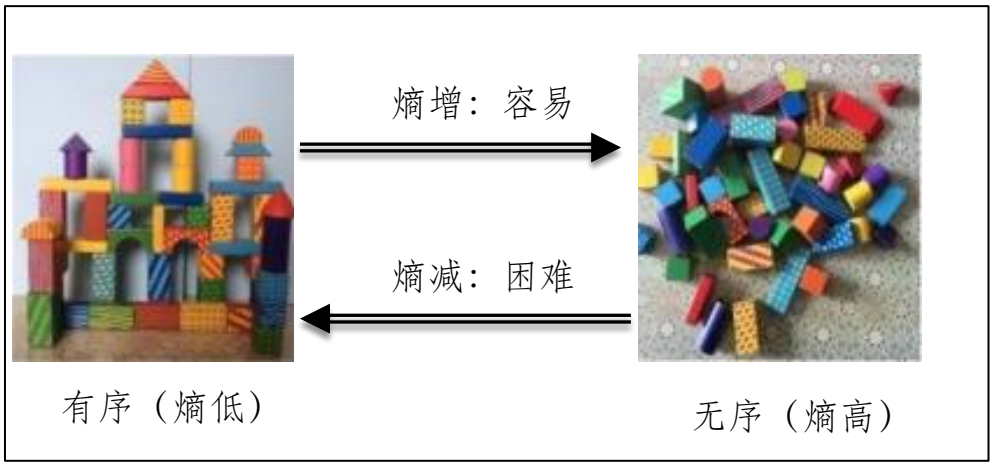



图 1.2 熵增容易熵减难

由此，我们可以从熵增定律得出以下三个结论：

1) 如果没有外部能量输入，封闭系统趋向越来越混乱（熵越来越大）。
2) 如果要让一个系统变得更有序，必须有外部能量的输入。
3) 当一个系统（或部分）变得更加有序，必然有另一个系统（或部分）变得更加无序，而且"无序"的增加程度将超过"有序"的增加程度。

从软件项目的代码编写层面看，代码质量代表着系统的有序程度，烂代码增加就是系统无序性上升的体现。每个不同的程序员编写的代码，甚至同一个程序员不同时期编写的代码，代码质量是无序的，随机的。在项目开发过程中，在没有外力影响的情况下，烂代码只会越来越多。为了维持系统有序，需要外界向系统不断输入能量。对于代码质量，我们需要主动投入资源，来有意识地抑制烂代码越来越多的自然趋势。

质量低的代码产生的常见原因是业务压力大，导致没有时间或意愿保障代码质量。因为向业务压力妥协而生产烂代码之后，开发效率会随之下降，导致业务压力更大，形成一种典型的恶性循环。

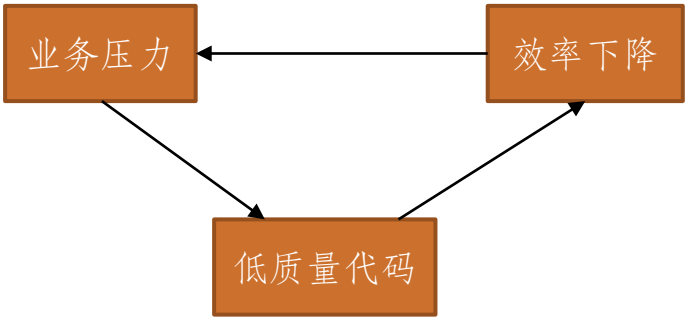



图 1.3 烂代码循环

在面临业务压力时，企业通常会倾向于通过增加人力来缓解业务压力。但从系统整体的角度来看，单纯地增加人力的话，会因为风格不一致、沟通成本上升等原因导致烂代码更多。人力增加会造成代码质量变差，开发效率下降，从而再度增大业务压力。这种代码质量越来越差的循环，是熵增定律在软件工程领域的生动体现。

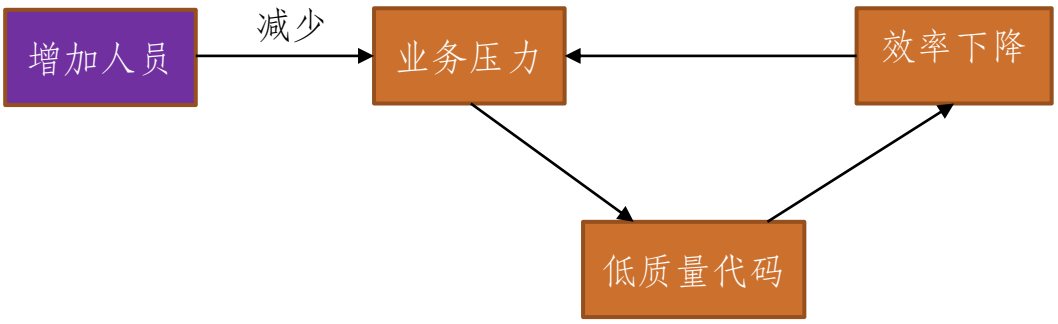



图 1.4 为项目增加人员

要遏制代码管控（Code Review）过程中的这种恶性循环，需要多管齐下，主动对代码质量进行管控，并且持续进行技术升级，系统性地解决问题。我们需要有意识地投入资源来建设代码质量的管控体系，这个体系的名称是 DevOps。

## 1.3研究目标

本项目的目标是探索研究适合 ABC 公司自身的软件代码成果管控体系模式，形成一套代码资产入库评测、安全运维、安全分享控制标准，开发一套满足该管控模式功能需要的软件平台，并通过实际软件开发项目验证。

1) 代码资产入库体系研究。从入库流程、代码的规范化要求、评审、测试、存储、安全运维等方面综合研究成果形成一套适合 ABC 公司自身环境的代码资产管控体系。

2) 代码出库及安全分享体系研究。研究不同形态代码资产的出库流程和方法，以及如何有效的保护知识产权以及安全泄密等问题，最终形成一套代码分享受控体系。

3) 代码资产管控平台开发。基于代码资产的入库出库体系和实施办法，开发出符合需求场景的软件工具平台，从而固化流程和规则并提高效率。

### 1.4研究内容

1) 代码规范研究。包括各类型开发语言框架的代码基础格式、组合形态规范，还要融入事件类代码规范化要求，例如代码内应包含运维所需的异常消息提醒代码等。

2) 代码资产入库评审流程和规则研究。包括评审专家团队的组建，如何发起评审，谁负责评审什么，评审的细粒度，如何对评审本身进行评价等。

3) 代码测试研究。面对复杂多样的代码成果，是否需要组织专业的测试团队，搭建测试工具和环境，并制定测试的输入输出要求、以及测试的覆盖面，如安全、性能、功能等在本阶段是否全覆盖以及如何覆盖。

4) 代码成果存储、运维研究。随着时间和项目的积累，代码及其说明文档的存储势必将有高要求，例如如何有序便捷存储，如何高效检索查阅。另外安全存放也是重点，是否由专业的运维团队支撑。

5) 代码分享安全受控办法研究。分享流程、规则、方法研究。

## 2 规划设计建议方案

### 2.1代码管控理论基础 DevOps

DevOps（Development 和 Operations 的组合词）是一组过程、方法与系统的统称，用于促进开发（应用程序/软件工程）、技术运营和质量保障（QA）部门之间的沟通、协作与整合。DevOps 可以提供高效稳定的、可持续的、可协调的、自动化的代码管控手段。

它是一种重视“软件开发人员（Dev）”和“IT 运维技术人员（Ops）”之间沟通合作的文化、运动或惯例。透过自动化“软件交付”和“架构变更”的流程，来使得构建、测试、发布软件能

够更加地快捷、频繁和可靠。它的出现是由于软件行业日益清晰地认识到：为了按时交付软件产品和服务，开发和运营工作必须紧密合作。

传统的软件组织将开发、IT 运营和质量保障设为各自分离的部门。在这种环境下如何采用新的开发方法（例如敏捷软件开发），这是一个重要的课题：按照从前的工作方式，开发和部署不需要 IT 支持或者 QA 深入的、跨部门的支持，而却需要极其紧密的多部门协作。然而 DevOps 考虑的还不止是软件部署。它是一套针对这几个部门间沟通与协作问题的流程和方法。

需要频繁交付的项目可能更需要项目团队有 DevOps 能力，使之能够支撑业务部门“每天部署 10 次”的要求—如果一个组织要生产面向多种用户、具备多样功能的应用程序，其部署周期必然会很短。这种能力也被称为持续部署，并且经常与精益创业方法联系起来。

DevOps 的引入能对产品交付、测试、功能开发和维护（包括曾经罕见但如今已屡见不鲜的“热补丁”）起到意义深远的影响。在缺乏 DevOps 能力的组织中，开发与运营之间存在着信息“鸿沟”—例如运营人员要求更好的可靠性和安全性，开发人员则希望基础设施响应更快，而业务用户的需求则是更快地将更多的特性发布给最终用户使用。这种信息鸿沟就是最常出问题的地方。

DevOps 希望做到的是软件产品交付过程中 IT 工具链的打通，使得各个团队减少时间损耗，更加高效地协同工作。专家们总结出了下面这个 DevOps 能力环，良好的闭环可以大大增加整体的产出。


图 2.1 DevOps 能力环

### 2.1.1 DevOps 的应用领域

DevOps 经常被描述为“开发团队与运营团队之间更具协作性、更高效的关系”。由于团队间协作关系的改善，整个组织的效率因此得到提升，伴随频繁变化而来的生产环境的风险也能得到降低。迫切需要使用 DevOps 的应用领域通常有:

- 使用敏捷或其他软件开发过程与方法
- 业务负责人要求加快产品交付的速率
- 虚拟化和云计算基础设施（可能来自内部或外部供应商）日益普遍
- 数据中心自动化技术和配置管理工具的普及

### 2.1.2 DevOps 对应用程序发布的影响

在企业中，应用程序发布是一项涉及多个团队、压力很大、风险很高的活动。然而在具备 DevOps 能力的组织中，应用程序发布的风险很低，原因如下：

（1）减少变更范围

与传统的瀑布式开发模型相比，采用敏捷或迭代式开发意味着更频繁的发布、每次发布包含的变化更少。由于部署经常进行，因此每次部署不会对生产系统造成巨大影响，应用程序会以平滑的速率逐渐生长。

（2）加强发布协调

靠强有力的发布协调人来弥合开发与运营之间的技能鸿沟和沟通鸿沟；采用电子数据表、电话会议、即时消息、企业门户等协作工具来确保所有相关人员理解变更的内容并全力合作。

（3）自动化

强大的部署自动化手段确保部署任务的可重复性、减少部署出错的可能性。与传统开发方法那种大规模的、不频繁的发布（通常以“季度”或“年”为单位）相比，敏捷方法大大提升了发布频率（通常以“天”或“周”为单位）。

### 2.1.3 DevOps 带来的好处

- 更小、更频繁的变更──意味着更少的风险
- 让开发人员更多地控制生产环境
- 更多地以应用程序为中心来理解基础设施
- 定义简洁明了的流程
- 尽可能地自动化

- 促成开发与运营的协作

## 2.2两大 DevOps 平台

现阶段有两个主要的商用 DevOps 平台，分别是微软 Azure DevOps 和 IBM DevOps Services。

### 2.2.1 Microsoft Azure DevOps

微软于 2018 年宣布推出 Azure DevOps，它是 Visual Studio Team Services 的继任者，以帮助开发人员更快地交付软件并提供更高质量的软件。DevOps 汇集人员、流程和技术，实现软件交付自动化，为用户提供持续的价值。借助 Azure DevOps 解决方案，无论 IT 部门有多大规模或使用何种工具，都可以更加快速可靠地交付软件。

开发者要使用 Azure DevOps 必须先注册和申请 Azure 用户，并创建 Azure DevOps 项目，Azure DevOps 项目提供一种简化的体验，开发者在其中既可使用现有的代码和 Git 存储库，也可选择一个示例应用程序，以便创建连接到 Azure 的持续集成 (CI) 和持续交付 (CD) 管道。

Azure DevOps 可以部署到 Azure 虚拟机，部署到 Azure SQL 数据库，部署到 Azure Kubernetes 服务，部署到 Azure Service Fabric。它提供以下功能：

- 使用 DevOps Projects 创建 CI/CD 管道
- 配置对 GitHub 存储库的访问权限并选择一个框架
- 配置 Azure DevOps 和 Azure 订阅
- 提交对 GitHub 所做的更改并将其自动部署到 Azure
- 检查 Azure Pipelines CI/CD 管道
- 配置 Azure Application Insights 监视
- 清理资源

借助 Azure 应用服务，你可以快速轻松地使用 Java、Node、PHP 或 ASP.NET 来创建 Web 应用，并使用 Docker 支持自定义语言运行时。持续集成和持续部署 (CI/CD) 管道可将每一个更改自动推送到 Azure 应用服务，让你更快地为客户创造价值。

使用容器，可轻松地持续生成和部署应用程序。使用 Azure Kubernetes 服务 (AKS) 协调这些容器的部署，获得可复制、可管理的容器群集。通过设置持续生成来生成容器映像和业务流程，可更快更可靠地进行部署。下图说明了一个典型的使用 Jenkins 和 Kubernetes 实现容器 CI/CD 样例：

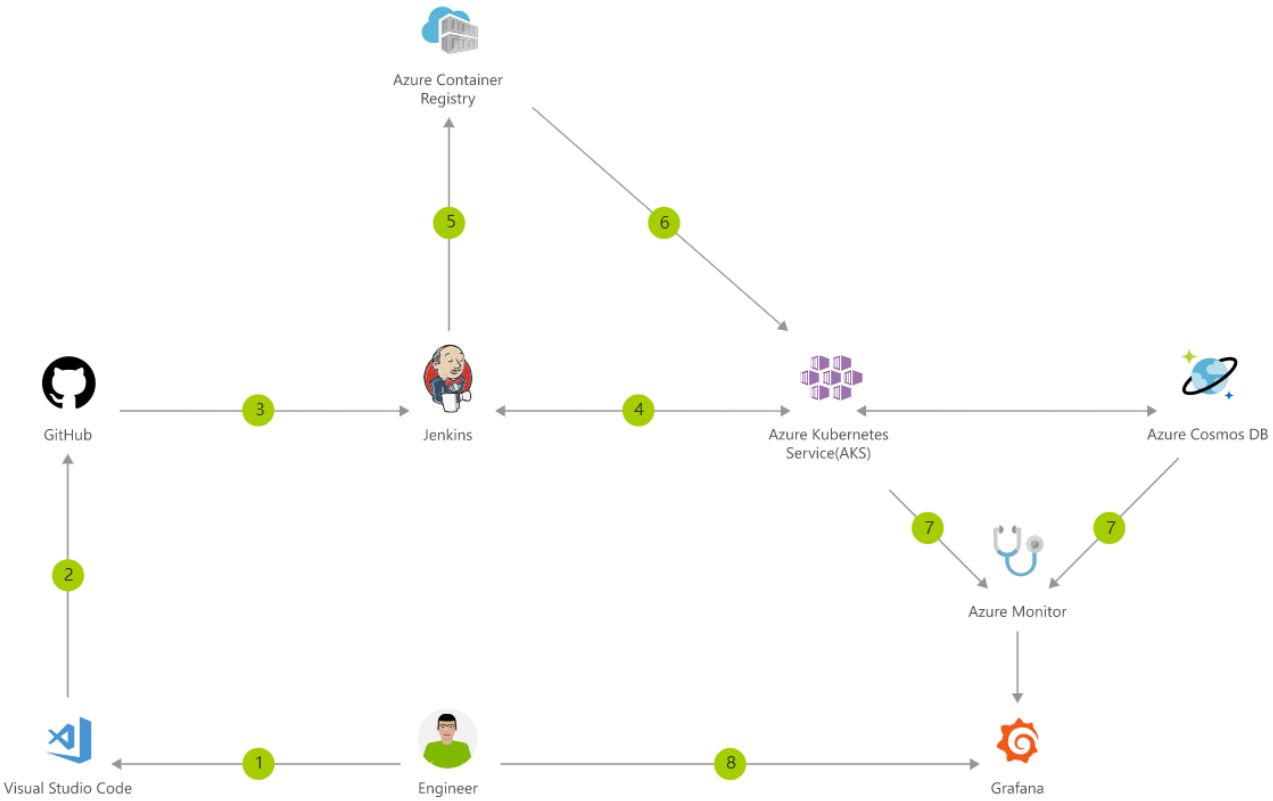



图 2.2 使用 Jenkins 和 Kubernetes 实现容器 CI/CD

图中的各阶段如下:

1. 更改应用程序源代码。

2. 将代码提交到 GitHub。

3. Jenkins 持续集成触发器。

4. Jenkins 触发生成作业，将 Azure Kubernetes 服务 (AKS) 用作动态生成代理。

5. Jenkins 生成 Docker 容器并将其推送到 Azure 容器注册表。

6. Jenkins 将新的容器化应用部署到 Kubernetes on Azure。

7. 容器服务 (AKS)，由 Azure Cosmos DB 支持。

8. Grafana 通过 Azure Monitor 显示经过可视化处理的基础架构和应用程序指标。

9. 监视应用程序并进行改进。

### 2.2.2 IBM Jazz

Jazz 是 IBM Rational 面向软件交付技术的下一代协作平台，是一个可扩充、可扩展的团队协作平台，可以无缝整合软件与系统开发生命周期中的工作。Jazz 旨在推动广大软件开发人员考虑摒弃单人开发工具，转而采用多人协作开发工具，以实现各人员间步调更一致的沟通与协作。

Jazz 使用一种名为“开放商业软件开发”的新形式进行开发。在传统商业开发流程中，新产品或新版本发布前，客户基本上无法了解产品的情况。与此不同的是，Jazz 的开发工作在 Jazz.net 以开放的方式进行。这种开放性和透明性的好处在于，它允许客户成为持续反馈循环的一部分，以便推动开发决策。您可以通过 Jazz.net 了解开发工作的进展情况，并可以下载 Jazz 最新的构建版本，亲自体验 Jazz 带给您及您团队的协作开发新体验。

Jazz 平台包括以下 Rational 工具：

- Rational Team Concert
- Rational Quality Manager
- Rational Requirements Composer
- Rational DOORS Next Generation
- Rational Software Architect Design Manager
- Rational Rhapsody Design Manager

Jazz 是面向生命周期协作的开放服务 OSLC (Open Services for Lifecycle Collaboration），Jazz 的优势有以下三点：

- Jazz 技术改变协作构建软件的方式，使软件交付更加协作化和高产。
- Jazz 技术使组织能针对各个项目和团队的需求采用合适规模的监管
- Jazz 的开放可扩展架构将使团队能组装专用的软件交付平台，并自行选择产品和解决方案。

JAZZ 实现 OSLC 见下图：

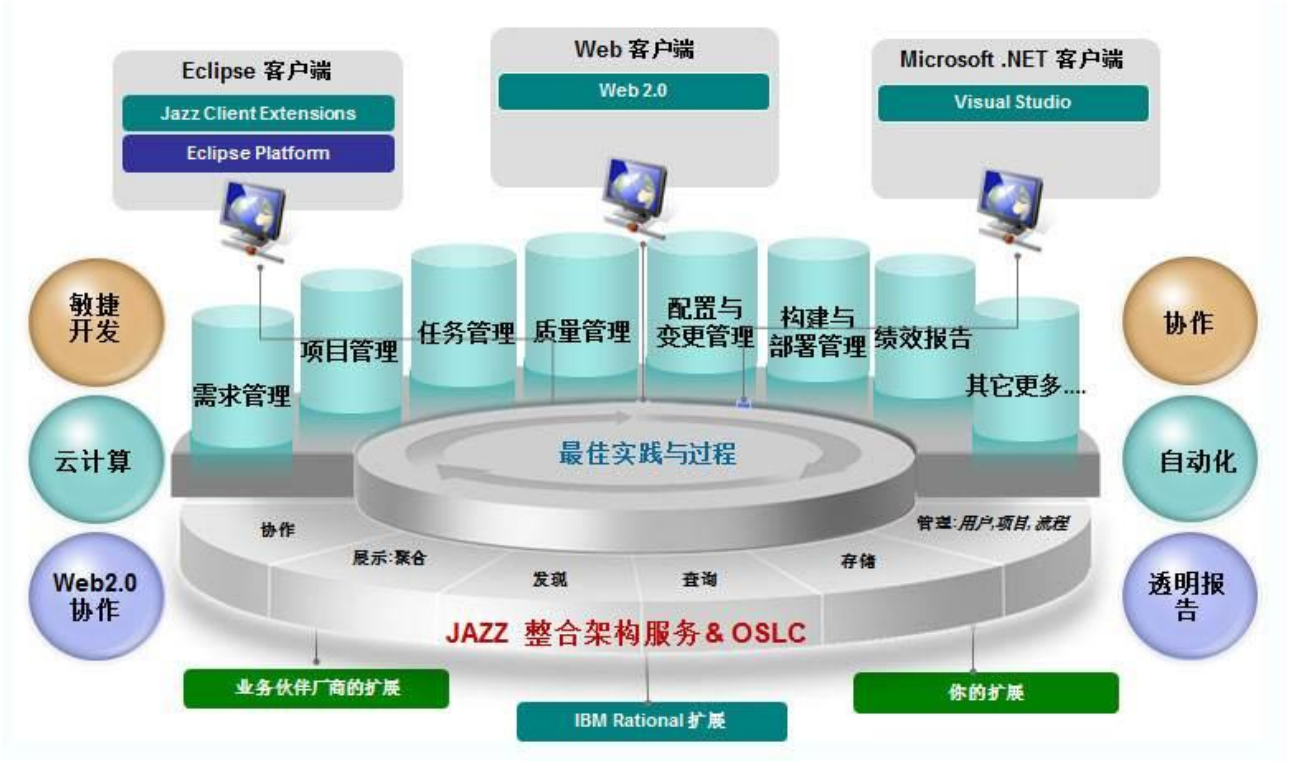



图 2.3 Jazz 整合架构服务和 OSLC

### 2.2.3 存在的问题

虽然 Microsoft Azure DevOps 和 IBM Jazz 都是重要的 DevOps 平台，功能也很强大，但两套系统并不兼容，对于 ABC 公司来说，不能选择这两个平台作为开发和代码管控平台。主要的原因如下:

- 系统过于复杂，模块过多，软件开发者和使用者都不容易掌握，如果要使用这些系统，会付出高昂的培训费用。
- 不能在本公司内部搭建 Azure DevOps 和 IDB Jazz 平台，有代码泄露的风险。
- 使用成本高。两个系统都需要支付用户费用。以 Azure DevOps 为例，Azure DevOps 对于开源项目和小项目（最多五个用户）是免费的。但是对于大型团队，费用从每月 30 美元（10 个用户）到每月 6,150 美元（1,000 个用户）。

综上所述，我们必须要选择可靠、易用、费用低廉并且能自我维护的代码管控平台。

## 2.3技术选型

ABC 公司原来也做过代码评审，管控，使用了一些建模工具，如 Visio、Rational Rose、PowerDesigner 等。这些工具只是单一的建模，不能进行代码管控，不能进行项目生命周期控制，代码更不能上网共享，因此费时费力，效率低下。

### 2.3.1 协议选型-Git

目前用到最广泛的版本控制器就是 SVN 和 Git，SVN（Subversion）是集中式管理的版本控制器，而 Git 是分布式管理的版本控制器！这是两者之间最核心的区别。

SVN 只有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。

- 每个版本库有唯一的 URL（官方地址），每个用户都从这个地址获取代码和数据；
- 获取代码的更新，也只能连接到这个唯一的版本库，同步以取得最新数据；
- 提交必须有网络连接（非本地版本库）；
- 提交需要授权，如果没有写权限，提交会失败；
- 提交并非每次都能够成功。如果有其他人先于你提交，会提示“改动基于过时的版本，先更新再提交”…诸如此类；
- 冲突解决是一个提交速度的竞赛：手快者，先提交，平安无事；手慢者，后提交，可能遇到麻烦的冲突解决。

Git 属于分布式的版本控制系统，它的速度飞快，极其适合管理大项目，它还有着令人难以置信的非线性分支管理系统，可以应付各种复杂的项目开发需求。Git 每一个终端都是一个仓库，客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。每一次的提取操作，实际上都是一次对代码仓库的完整备份。

与 SVN 不同，Git 记录版本历史只关心文件数据的整体是否发生变化。Git 并不保存文件内容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连接。简略的说，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。

本项目选择 Git，原因有：

- Git 分支功能最为强大，分支管理能力让 SVN 望尘莫及。Git 可以很容易地对比两个分支，知道一个分支中哪些提交尚未合并到另一分支，反之亦然。

查看当前分支比 other 分支多了哪些提交：

```
$ git log other ..
```

查看 other 分支比当前分支多了哪些提交：

```
$ git log ..other
```

SVN 的分支并不是真正的分支，因为分支最基本的提交隔离 SVN 就没能实现。 在 SVN 中一次提交可以同时更改主线（/trunk）和分支中的内容， 所以 SVN 不能判断一个分支中哪些提交未合并到另外的分支。

- Git 可以实现更好的发布控制

针对同一个项目，Git 可以设置不同层级的版本库（多版本库）， 或者通过不同的分支（多分支）实现对发布的控制。设置只有发布管理员才有权限推送的版本库或者分支，用于稳定发布版本的维护。设置只有项目经理、模块管理员才有权推送的版本库或者分支，用用于整合测试。

- 隔离开发，提交审核

如何对团队中的新成员的开发进行审核呢？在 Git 服务器上可以实现用户自建分支和自建版本库的功能， 这样团队中的新成员既能将本地提交推送到服务器以对工作进行备份，又能够方便团队中的其他成员对自己的提交进行审核。

审核新成员提交时，从其个人版本库或个人分支获取（fetch）提交，从提交说明、代码规范、编译测试等多方面对提交逐一审核。审核通过执行 git merge 命令合并到开发主线中。

- 对合并更好的支持，更少的冲突，更好的冲突解决

因为 Git 基于对内容的追踪而非对文件名追踪，所以遇到一方或双方对文件名更改时，Git 能够很好进行自动合并或提供工具辅助合并。而 SVN 遇到同样问题时会产生树冲突，解决起来很麻烦。

Git 的基于 DAG（有向非环图）的设计比 SVN 的线性提交提供更好的合并追踪， 避免不必要的冲突，提高工作效率。这是开发者选择 Git、抛弃 SVN 的重要理由。

- 保证已修复 Bug 不再重现

以为创建完毕里程碑标签（tag）便完成软件版本的发布是有风险的，往往会由于之前的版本（维护版本）中的一些 Hotfix 提交没有合并到最新版本而造成已修复问题在新版本中重现。

Git 分支和合并追踪可以解决这个问题。例如用 maint 分支跟踪最新的发行版，当确定里程碑 tag v1.6.4 为最新发行版时，在 maint 分支执行如下命令以切换到最新发行版：

```
$ git checkout maint
$ git merge --ff-only v1.6.4
```

如果合并成功，代表发行版 v1.6.4 包含了所有前一个发行版的提交。反之说明前一个发行版某个或某些 Hotfix 提交尚未合并到最新发行版中。

- 版本库的安全性

SVN 版本库安全性很差，是管理员头痛的问题。SVN 版本库服务器端历史数据被篡改，或者硬盘故障导致历史数据被篡改时，客户端很难发现。管理员的备份也会被污染。SVN 作为集中式版本控制系统，存在单点故障的风险。备份版本库的任务非常繁重。

Git 在这方面完胜 SVN。首先 Git 是分布式版本控制系统，每个用户都相当于一份备份，管理员无需为数据备份而担心。再有 Git 中包括提交、文件内容等都通过计算 SHA1 哈希值保证数据的完整性，任何恶意篡改历史数据都会被及时发现从而被挫败。

### 2.3.2 平台选型-GitLab

国外信息化发达国家如美国在这方面的认识相对比国内更早，例如 Github 是目前软件代码仓库托管存放最多的工具平台，在代码管理方面还有较多像 Gogs、Phabricator 这样优秀的开源工具。在国内，也有一个著名的平台：码云 gitee，以其极速、稳定，更友好易用的特点受到好评。除了 GitLab，其他现行主要代码管控平台的优劣对比表如下：

表 2.1 平台选型对比表

| 架构名称 | GitLab | Github | Gogs | Phabricator | 码云 gitee |
|---|---|---|---|---|---|
| 概述 | 免费的、开源的、分布式的版本控制系统。 | 全球最大的代码托管平台。 | Google 的开源代码管理工具 | Facebook 的软件开发管理和代码管理评审工具，开源 | 国内专业、强大的企业级代码托管服务平台 |

| | | | | | |
|---|---|---|---|---|---|
| 是否支持企业内部自主搭建 | 支持 | 不支持 | 支持 | 支持 | 不支持 |
| 是否支持质量控制 | 支持 | 支持 | 支持 | 支持 | 支持 |
| 是否支持讨论和协作 | 支持 | 支持 | 不支持 | 支持 | 支持 |
| 代码片段分享 | 支持 | 支持 | 不支持 | 支持 | 不支持 |
| 主流厂商支持 | IBM，Sony，NASA，Alibaba | Microsoft | Google，igt | Facebook | 深圳市奥思网络科技有限公司 |
| 使用规模 | 未公开<br>估计至少是千万以上的开发人员使用。 | 2800 万开发人员正在使用，拥有 7900 万代码库。 | 未公开 | 未公开 | 超过 300 万的开发者， 50,000 家公司和机构 |
| 优劣对比总结 | 支持企业内部自主搭建，功能全面，代码托管，操作非常快速、简便。支持“持续集成” | 功能全面，但不开源，不支持私有化部署 | 轻量开源，支持私有化部署管理，在代码评审方面不强 | 功能强大且开源可私有化，可扩展性强 | 不开源，代码托管，在线 IDE，企业级研发协作，敏捷开发管理，Wiki |

根据前期评估，本项目选择 GitLab 作为代码管控平台。最主要的理由就是 GitLab 支持企业内部自主搭建，功能全面，成熟。

GitLab 在 2011 年推出，和 GitHub 一样，是基于网络的 Git 仓库，截至 2018 年 5 月，该公司约有 290 名团队成员，另外还有 2000 多名开源贡献者，IBM、Sony、NASA 等公司和组织都有使用。

GitLab 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。

GitLab 是一个用于仓库管理系统的开源项目，使用 Git 作为代码管理工具，并在此基础上搭建起来的 Web 服务。Gitlab 是一个用 Ruby on Rails 开发的开源项目管理程序，可以通过 WEB 界面进行访问公开的或者私人项目。它能够浏览源代码，管理缺陷和注释。

GitLab 具有完整的分支管理和用户管理功能，能够完成各种项目的源代码及版本的管控。其核心功能包括:

- 软件项目（资料库）管理
- 标签管理
- 分配成员角色;
- 锁定受保护的分支;
- 发起、审查、处理合并请求;
- 代码评论
- 代码片断管理
- 邮件通知
- 持续集成

除此之外，GitLab 还是一个开源系统，开发接口，允许第三方软件接入。

本项目采用 GitLab 可以很方便地将本公司内部的管理人员与第三方软件公司的程序人员整合在一个平台上，各自扮演自己的角色，最后完成代码成果规范、代码成果入库质量评审、检测、代码成果存储、运维以及代码成果出库等既定目标的工作。

基于对 ABC 集团公司软件代码管控需求以及现有流程的调研，我们设计了三种代码管控模型：标准模型，精简模型以及开发商高级模型。

- 标准模型主要针对在研的大中型项目的代码管控;
- 精简模型主要针对已完成项目或者小型项目的代码管控。
- 开发商高级模型主要描述开发商怎样使用 GitLab 平台的方法。

### 2.3.3 GitLab 实现 DevOps 生命周期

GitLab是第一个支持Concurrent DevOps的软件开发，安全和运营的单一应用程序平台，可以加快软件生命周期并从根本上提高业务速度。GitLab 为 DevOps 生命周期的所有阶段提供解决方案，如下表所示：


表 2.2 GitLab 的 DevOps 生命周期表

| DevOps 阶段 | 功能 |
| --- | --- |
| 管理 | 统计和分析功能。<br>GitLab 提供统计数据和洞察力，最大限度地提高组织中 GitLab 的价值。 |
| 规划 | 项目规划和管理功能。<br>无论对瀑布模型还是敏捷模型，GitLab 都可以简化协作工作流程。使用 GitLab 灵活的项目管理工具，以自己的方式可视化，确定优先级，协调和跟踪项目进度。 |
| 创建 | 源代码和数据创建和管理功能。<br>将源代码整合到一个易于管理和控制的分布式版本控制系统中，而不会中断工作流程。GitLab 的 Git 存储库配备了分支工具和访问控制，为项目和代码的协作提供了可扩展的单一事实来源。 |
| 检验 | 测试，代码质量和持续集成功能。<br>通过内置静态代码分析，代码测试，代码质量，依赖性检查和评论应用程序，更快地发现错误，提高安全性并缩短反馈周期。自定义批准工作流程控件，自动测试代码质量，并为每个代码更改启动临时环境。GitLab 持续集成是最受欢迎的下一代测试系统，可以扩展以更快地运行测试。 |
| 包 | Docker 容器注册表。<br>GitLab 容器注册表提供增强的自定义 Docker 镜像的安全性和访问控制，而无需第三方加载项。通过完整的 Git 存储库管理集成轻松上传和下载 GitLab CI / CD 中的映像。 |
| 发布 | 应用程序发布和交付功能。<br>花费更少的时间配置您的工具，并花更多的时间创建。无论您是部署到一台服务器还是数千台，使用 GitLab 内置的持续交付和部署，可以放心，安全地构建，测试和发布您的代码。 |
| 配置 | 应用和基础架构配置工具。<br>使用 GitLab Auto DevOps 自动完成从构建到部署和监控的整个工作流程。最佳实践模板可让您从最小到零的配置开始。然后自定义从 buildpacks 到 CI / CD 的所有内容。 |

| | |
|---|---|
| 监控 | 应用程序监控和指标功能。<br>确保应用程序始终响应并可用。GitLab 收集并显示已部署应用程序的性能指标，以便用户可以立即了解代码更改如何影响生产环境。 |
| 安全 | 安全功能。<br>检查应用程序是否存在可能导致未经授权的访问，数据泄漏和拒绝服务的安全漏洞。GitLab 将对应用程序代码执行静态和动态测试，查找已知缺陷并在合并请求中报告它们，以便您可以在合并之前修复它们。安全团队可以使用仪表板获取项目和组的高级视图，并在需要时启动修复过程。 |

# 3 项目任务分步计划

根据项目主要研究内容，项目分为调研，技术选型及方案设计，平台开发，代码入库，工作流规范五个阶段，所下表所示：

表 3.1 项目任务分步计划表

| 时间 | 任务目标 |
|---|---|
| 3 个月<br>已完成 | 调研。<br>深度调研学习国内外本课题相关技术与实践经验，收集整理公司内部历史软件开发项目资料，并调研公司内部软件代码管控情况，评估调研报告。 |
| 8 个月<br>已完成 | 技术选型与方案设计。<br>设计代码资产管理系统、代码入库体系、代码成果安全分享体系设计，并评审。完成代码入库体系流程方案设计；完成代码成果安全分享体系方案设计。 |
| 10 个月<br>正在开发 | 平台开发。<br>通过调用 GitLab API 开发一套“代码资产管控平台”，配置 GitLab 平台，使代码入库，代码管控本地化，达到更快捷，更高效的目标。 |
| 6 个月<br>待办 | 代码入库。<br>将公司现有代码输入“代码资产管控平台”，公司正式进入代码管控阶段。解决了以下问题：<br>有哪些项目和代码。有序、便捷存储代码，高效检索查阅。<br>代码运行环境，如何发布代码。<br>是谁开发了代码。 |

| | |
|---|---|
| | 代码发布在哪里。<br>是谁使用代码。 |
| 6 个月<br>待办 | 工作流规范。<br>在公司原有代码入库完成之后，对新项目的开发过程进行全面的管控，应用效果前景如下：<br>软件开发项目的代码成果管控水平明显提高。<br>软件质量明显提高，故障更少。<br>效益显著提高。代码成果的复用率普遍提高，开发成本、迭代时间成本降低。<br>综合竞争力上明显提高。通过程序代码集中管理，有效保护知识产权，软件和专业结合更紧密，油气综合技术服务能力增强，同时降低数据信息安全风险。 |