(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 110865806 B
(45)授权公告日 2023.08.18

(21)申请号 201911142035.8

(22)申请日 2019.11.20

(65)同一申请的已公布的文献号
申请公布号 CN 110865806 A

(43)申请公布日 2020.03.06

(73)专利权人 腾讯科技(深圳)有限公司
地址 518057 广东省深圳市南山区高新区科技中一路腾讯大厦35层

(72)发明人 陈胜华

(74)专利代理机构 北京三高永信知识产权代理有限责任公司 11138
专利代理师 邢惠童

(51)Int.Cl.
G06F 8/30(2018.01)

(56)对比文件
CN 106201861 A,2016.12.07
CN 101706896 A,2010.05.12
CN 106202192 A,2016.12.07
CN 108182059 A,2018.06.19
CN 108415694 A,2018.08.17
CN 108536432 A,2018.09.14
CN 108536472 A,2018.09.14
CN 109032949 A,2018.12.18
CN 109101222 A,2018.12.28
CN 110069260 A,2019.07.30
CN 110322230 A,2019.10.11
US 2006247964 A1,2006.11.02
US 2017169223 A1,2017.06.15
US 2018068271 A1,2018.03.08
US 2018293058 A1,2018.10.11

审查员 吴银娥

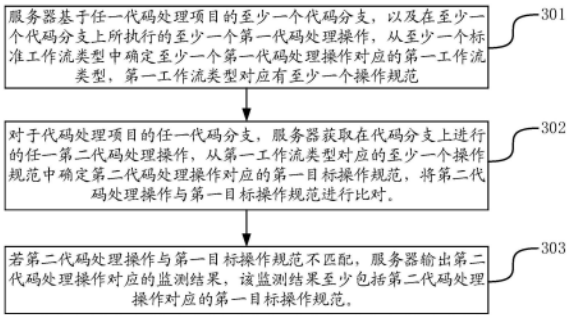
权利要求书3页 说明书23页 附图7页

(54)发明名称

代码处理方法、装置、服务器及存储介质

(57)摘要

本申请提供了一种代码处理方法、装置、服务器及存储介质，属于计算机技术领域。所述方法包括：基于任一代码处理项目的至少一个代码分支，以及在代码分支上所执行的第一代码处理操作，从至少一个标准工作流类型中确定第一工作流类型，第一工作流类型对应有至少一个操作规范；对于代码处理项目的任一代码分支，获取在代码分支上进行的任一第二代码处理操作，从至少一个操作规范中确定第二代码处理操作对应的第一目标操作规范，将第二代码处理操作与第一目标操作规范进行比对；若第二代码处理操作与第一目标操作规范不匹配，输出监测结果。在本申请中，服务器能够辅助开发团队更加准确、高效的应用标准工作流，使开发团队协作开发的效率更高。



CN 110865806 B

1.一种代码处理方法,其特征在于,标准工作流类型包括主干开发工作流、Git Flow工作流、GitHub工作流、GitLab环境分支工作流和GitLab发布分支工作流,所述方法包括:

当默认分支上的代码提交的频繁度大于预设阈值时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为主干开发工作流,所述第一工作流类型对应有至少一个操作规范;

当永久分支的数量为第一数值时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitLab环境分支工作流;

当所述永久分支的数量为第二数值时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为Git Flow工作流;

当所述永久分支的数量为第三数值,且所述发布分支和所述默认分支为同一个代码分支时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitHub工作流;

当所述永久分支的数量为第三数值,且所述发布分支和所述默认分支为不同的代码分支时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitLab发布分支工作流;

对于代码处理项目的任一代码分支,获取在所述代码分支上进行的任一第二代码处理操作,从所述第一工作流类型对应的所述至少一个操作规范中确定所述第二代码处理操作对应的第一目标操作规范,将所述第二代码处理操作与所述第一目标操作规范进行比对;

若所述第二代码处理操作与所述第一目标操作规范不匹配,输出所述第二代码处理操作对应的监测结果,所述监测结果至少包括所述第一目标操作规范。

2.根据权利要求1所述的方法,其特征在于,所述将所述第二代码处理操作与所述第一目标操作规范进行比对,包括:

根据所述第二代码处理操作,确定所述第二代码处理操作对应的第一代码分支;

根据所述第一代码分支上进行所述第二代码处理操作前后的代码,确定所述第一代码分支的分支变化;

将所述第一代码分支的分支变化与所述第一目标操作规范所确定的分支变化信息进行比对。

3.根据权利要求1所述的方法,其特征在于,所述方法还包括:

基于任一代码处理项目的至少一个代码分支,以及在所述至少一个代码分支上所执行的至少一个第三代码处理操作,将所述至少一个第三代码处理操作输入到机器学习模型中,输出第三工作流类型,所述第三工作流类型对应有至少一个操作规范;

将所述第三工作流类型确定为所述至少一个标准工作流类型中的一个。

4.根据权利要求1所述的方法,其特征在于,所述从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型之后,所述方法还包括:

对于所述至少一个第一代码处理操作中的每一个第一代码处理操作,从所述第一工作流类型对应的所述至少一个操作规范中确定所述第一代码处理操作对应的第二目标操作规范,将所述第一代码处理操作与所述第二目标操作规范进行比对;

若所述第一代码处理操作与所述第二目标操作规范不匹配,根据所述第一代码处理操作与所述第二目标操作规范,确定所述第一代码处理操作对应的比对结果。

5.一种代码处理装置,其特征在于,标准工作流类型包括主干开发工作流、Git Flow工作流、GitHub工作流、GitLab环境分支工作流和GitLab发布分支工作流,所述装置包括:

确定模块,被配置为当默认分支上的代码提交的频繁度大于预设阈值时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为主干开发工作流,所述第一工作流类型对应有至少一个操作规范;当永久分支的数量为第一数值时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitLab环境分支工作流;当所述永久分支的数量为第二数值时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为Git Flow工作流;当所述永久分支的数量为第三数值,且所述发布分支和所述默认分支为同一个代码分支时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitHub工作流;当所述永久分支的数量为第三数值,且所述发布分支和所述默认分支为不同的代码分支时,从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitLab发布分支工作流;

比对模块,被配置为对于代码处理项目的任一代码分支,获取在所述代码分支上进行的任一第二代码处理操作,从所述第一工作流类型对应的所述至少一个操作规范中确定所述第二代码处理操作对应的第一目标操作规范,将所述第二代码处理操作与所述第一目标操作规范进行比对;

输出模块,被配置为若所述第二代码处理操作与所述第一目标操作规范不匹配,输出所述第二代码处理操作对应的监测结果,所述监测结果至少包括所述第一目标操作规范。

6.根据权利要求5所述的装置,其特征在于,所述装置还包括:

机器学习模块,被配置为基于任一代码处理项目的至少一个代码分支,以及在所述至少一个代码分支上所执行的至少一个第三代码处理操作,将所述至少一个第三代码处理操作输入到机器学习模型中,输出第三工作流类型,所述第三工作流类型对应有至少一个操作规范;

所述确定模块,还被配置为将所述第三工作流类型确定为所述至少一个标准工作流类型中的一个。

7.根据权利要求5所述的装置,其特征在于,所述装置还包括:

所述比对模块,还被配置为对于所述至少一个第一代码处理操作中的每一个第一代码处理操作,从所述第一工作流类型对应的所述至少一个操作规范中确定所述第一代码处理操作对应的第二目标操作规范,将所述第一代码处理操作与所述第二目标操作规范进行比对;

所述确定模块,还被配置为若所述第一代码处理操作与所述第二目标操作规范不匹配,根据所述第一代码处理操作与所述第二目标操作规范,确定所述第一代码处理操作对应的比对结果。

8.一种服务器,其特征在于,所述服务器包括处理器和存储器,所述存储器中存储有至少一条程序代码,所述至少一条程序代码由所述处理器加载并执行,以实现如权利要求1-4任一项所述的代码处理方法。

9.一种计算机可读存储介质,其特征在于,所述计算机可读存储介质中存储有至少一条程序代码,所述至少一条程序代码由处理器加载并执行,以实现如权利要求1-4任一项所

述的代码处理方法。

代码处理方法、装置、服务器及存储介质

技术领域

[0001] 本申请涉及计算机技术领域，特别涉及一种代码处理方法、装置、服务器及存储介质。

背景技术

[0002] 软件开发需要整个开发团队的协作，开发团队会创建代码分支，在相应的代码分支上进行代码处理。为了使软件的版本发布更加可靠，需要维护好代码分支。

[0003] 标准工作流提供了标准的代码分支维护方案。例如，标准工作流对应有对代码分支的创建、代码分支的合并和代码的分支销毁等方面的操作规范。开发团队可以选定一种标准工作流进行代码处理。开发团队中的每一个成员都遵守选定的标准工作流对应的操作规范，才能维护好代码分支，保证团队协作开发的效率。

[0004] 当前，在开发团队指定一种标准工作流后，开发团队中的成员仅能依靠自己对该标准工作流对应的操作规范的理解来进行代码处理，难以保证开发团队中的每个成员都能严格遵守选定的标准工作流对应的操作规范，从而不能保证团队协作开发的效率。

发明内容

[0005] 本申请实施例提供了一种代码处理方法、装置、服务器及存储介质，能够解决团队协作开发效率较低的问题。所述技术方案如下：

[0006] 根据本申请实施例的一方面，提供了一种代码处理方法，所述方法包括：

[0007] 基于任一代码处理项目的至少一个代码分支，以及在所述至少一个代码分支上所执行的至少一个第一代码处理操作，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型，所述第一工作流类型对应有至少一个操作规范；

[0008] 对于所述代码处理项目的任一代码分支，获取在所述代码分支上进行的任一第二代码处理操作，从所述第一工作流类型对应的所述至少一个操作规范中确定所述第二代码处理操作对应的第一目标操作规范，将所述第二代码处理操作与所述第一目标操作规范进行比对；

[0009] 若所述第二代码处理操作与所述第一目标操作规范不匹配，输出所述第二代码处理操作对应的监测结果，所述监测结果至少包括所述第一目标操作规范。

[0010] 在一种可能的实现方式中，所述根据所述默认分支、所述发布分支、所述默认分支上的代码提交的频繁度和所述永久分支的数量中的至少一个，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型之前，所述方法还包括：

[0011] 当所述第一代码处理操作为代码发布操作时，从所述代码分支中确定所述代码发布操作对应的发布分支，所述代码发布操作用于在所述发布分支上进行代码发布；

[0012] 当所述第一代码处理操作为代码提交操作时，根据所述代码提交操作，确定所述默认分支上的代码提交的频繁度；

[0013] 从所述代码分支中确定所述永久分支的数量。

[0014] 在另一种可能的实现方式中，所述若所述第二代码处理操作与所述第一目标操作规范不匹配，输出监测结果，包括：

[0015] 若所述第二代码处理操作与所述第一目标操作规范不匹配，生成所述第二代码处理操作对应的监测结果，以及，若所述第二代码处理操作与所述第一目标操作规范不匹配，根据所述第二代码处理操作和所述第一目标操作规范，对所述第二代码处理操作对应的第二代码分支执行第四代码处理操作；

[0016] 输出所述监测结果。

[0017] 根据本申请实施例的另一方面，提供了一种代码处理装置，所述装置包括：

[0018] 确定模块，被配置为基于任一代码处理项目的至少一个代码分支，以及在所述至少一个代码分支上所执行的至少一个第一代码处理操作，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型，所述第一工作流类型对应有至少一个操作规范；

[0019] 比对模块，被配置为对于所述代码处理项目的任一代码分支，获取在所述代码分支上进行的任一第二代码处理操作，从所述第一工作流类型对应的所述至少一个操作规范中确定所述第二代码处理操作对应的第一目标操作规范，将所述第二代码处理操作与所述第一目标操作规范进行比对；

[0020] 输出模块，被配置为若所述第二代码处理操作与所述第一目标操作规范不匹配，输出所述第二代码处理操作对应的监测结果，所述监测结果至少包括所述第一目标操作规范。

[0021] 在一种可能的实现方式中，所述比对模块，还被配置为根据所述第二代码处理操作，确定所述第二代码处理操作对应的第一代码分支；根据所述第一代码分支上进行所述第二代码处理操作前后的代码，确定所述第一代码分支的分支变化；将所述第一代码分支的分支变化与所述第一目标操作规范所确定的分支变化信息进行比对。

[0022] 在另一种可能的实现方式中，所述代码分支包括默认分支、永久分支和发布分支；所述确定模块，还被配置为根据所述默认分支、所述发布分支、所述默认分支上的代码提交的频繁度和所述永久分支的数量中的至少一个，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型。

[0023] 在另一种可能的实现方式中，所述标准工作流类型包括主干开发工作流、Git Flow工作流、GitHub工作流、GitLab环境分支工作流和GitLab发布分支工作流；所述确定模块，还被配置为当所述默认分支上的代码提交的频繁度大于预设阈值时，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为主干开发工作流；当所述永久分支的数量为第一数值时，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitLab环境分支工作流；当所述永久分支的数量为第二数值时，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为Git Flow工作流；当所述永久分支的数量为第三数值，且所述发布分支和所述默认分支为同一个代码分支时，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitHub工作流；当所述永久分支的数量为第三数值，且所述发布分支和所述默认分支为不同的代码分支时，从至少一个标准工作流类型中确定所述至少一个第一代码处理操作对应的第一工作流类型为GitLab发布分支工

作流。

[0024] 在另一种可能的实现方式中，所述确定模块，还被配置为当所述第一代码处理操作为代码发布操作时，从所述代码分支中确定所述代码发布操作对应的发布分支，所述代码发布操作用于在所述发布分支上进行代码发布；当所述第一代码处理操作为代码提交操作时，根据所述代码提交操作，确定所述默认分支上的代码提交的频繁度；从所述代码分支中确定所述永久分支的数量。

[0025] 在另一种可能的实现方式中，所述确定模块，还被配置为根据所述第一代码处理操作，确定所述第一代码处理操作对应的代码处理项目；确定所述代码处理项目对应的配置信息，所述配置信息包括所述代码处理项目对应的所述至少一个标准工作流类型中的第二工作流类型；将所述第二工作流类型确定为所述至少一个第一代码处理操作对应的第一工作流类型。

[0026] 在另一种可能的实现方式中，所述装置还包括：

[0027] 机器学习模块，被配置为基于任一代码处理项目的至少一个代码分支，以及在所述至少一个代码分支上所执行的至少一个第三代码处理操作，将所述至少一个第三代码处理操作输入到机器学习模型中，输出第三工作流类型，所述第三工作流类型对应有至少一个操作规范；

[0028] 所述确定模块，还被配置为将所述第三工作流类型确定为所述至少一个标准工作流类型中的一个。

[0029] 在另一种可能的实现方式中，所述输出模块，还被配置为若所述第二代码处理操作与所述第一目标操作规范不匹配，生成所述第二代码处理操作对应的监测结果，以及，若所述第二代码处理操作与所述第一目标操作规范不匹配，根据所述第二代码处理操作和所述第一目标操作规范，对所述第二代码处理操作对应的第二代码分支执行第四代码处理操作；输出所述监测结果。

[0030] 在另一种可能的实现方式中，所述装置还包括：

[0031] 所述比对模块，还被配置为对于所述至少一个第一代码处理操作中的每一个第一代码处理操作，从所述第一工作流类型对应的所述至少一个操作规范中确定所述第一代码处理操作对应的第二目标操作规范，将所述第一代码处理操作与所述第二目标操作规范进行比对；

[0032] 所述确定模块，还被配置为若所述第一代码处理操作与所述第二目标操作规范不匹配，根据所述第一代码处理操作与所述第二目标操作规范，确定所述第一代码处理操作对应的比对结果。

[0033] 根据本申请实施例的另一方面，提供了一种服务器，所述服务器包括处理器和存储器，所述存储器中存储有至少一条程序代码，所述至少一条程序代码由所述处理器加载并执行，以实现上述任一可能实现方式所述的代码处理方法。

[0034] 根据本申请实施例的另一方面，提供了一种计算机可读存储介质，所述计算机可读存储介质中存储有至少一条程序代码，所述至少一条程序代码由处理器加载并执行，以实现上述任一可能实现方式所述的代码处理方法。

[0035] 在本申请实施例中，对于任一代码处理项目的代码分支，服务器能够自动根据代码分支上所执行的第一代码处理操作，确定该代码处理项目对应的开发团队应用的第一工

作流类型;监测代码分支上进行的第二代码处理操作,与第一工作流类型对应的第一目标操作规范进行比对;对于与第一目标操作规范不匹配的第二代码处理操作,输出相应的监测结果,监测结果中至少包括第一目标操作规范。服务器能够确定任一代码处理项目对应的第一工作流类型,基于该第一工作流类型对应的操作规范,监测代码分支上执行的代码处理操作;对于未按照操作规范执行的代码处理操作,服务器能够输出至少包括第一目标操作规范的监测结果,从而使得开发团队的成员能够基于该监测结果对未按照操作规范执行的代码处理操作及时进行修正。在本申请中,服务器能够辅助开发团队更加准确、高效的应用标准工作流,使开发团队协作开发的效率更高。

附图说明

[0036] 为了更清楚地说明本申请实施例中的技术方案,下面将对实施例描述中所需要使用的附图作简单地介绍,显而易见地,下面描述中的附图仅仅是本申请的一些实施例,对于本领域普通技术人员来讲,在不付出创造性劳动的前提下,还可以根据这些附图获得其他的附图。

[0037] 图1是本申请实施例提供的一种实施环境的示意图;

[0038] 图2是本申请实施例提供的一种代码处理方法的示意图;

[0039] 图3是本申请实施例提供的一种代码处理方法的流程图;

[0040] 图4是本申请实施例提供的一种“上游优先原则”的示意图;

[0041] 图5是本申请实施例提供的一种确定第一工作流类型的流程图;

[0042] 图6是本申请实施例提供的一种Git Flow工作流的分支模型的示意图;

[0043] 图7是本申请实施例提供的一种GitHub工作流的分支模型的示意图;

[0044] 图8是本申请实施例提供的一种GitLab环境分支工作流的分支模型的示意图;

[0045] 图9是本申请实施例提供的一种GitLab发布分支工作流的分支模型的示意图;

[0046] 图10是本申请实施例提供的一种主干开发工作流的分支模型的示意图;

[0047] 图11是本申请实施例提供的一种代码处理装置的框图;

[0048] 图12是本申请实施例提供的一种服务器的框图。

具体实施方式

[0049] 为使本申请的目的、技术方案和优点更加清楚,下面将结合附图对本申请实施方式作进一步地详细描述。

[0050] 本申请的说明书和权利要求书及所述附图中的术语“第一”、“第二”、“第三”和“第四”等是用于区别不同对象,而不是用于描述特定顺序。此外,术语“包括”和“具有”以及它们的任意变形,意图在于覆盖不排他的包含。例如包含了一系列步骤或单元的过程、方法、系统、产品或设备没有限定于已列出的步骤或单元,而是可选地还包括没有列出的步骤或单元,或可选地还包括对于这些过程、方法、产品或设备固有的其他步骤或单元。

[0051] 代码是软件开发的核心资产,由于软件开发过程较长,并且需要多人同时协作开发,软件开发的开发团队通常会选择一个可靠的代码处理工具来进行代码管理和版本控制。例如,该可靠的代码处理工具可以为Git(一种开源的分布式版本控制系统)。

[0052] 对于一个软件开发项目,开发团队可以创建一个代码处理项目进行代码管理,开

发团队的成员可以在其所属的开发团队的代码处理项目中通过Git创建代码分支，在不同的代码分支上进行代码的开发、版本的发布上线、缺陷的修复等，实现代码的隔离。例如，代码分支可以为Git分支，代码分支包括特性分支、发布分支和缺陷修复分支等。其中，特性分支用于进行代码的开发，发布分支用于进行版本的发布上线，缺陷修复分支用于进行缺陷的修复。开发团队的成员在特性分支上完成代码的开发或者在缺陷修复分支上完成缺陷的修复后，可以将特性分支或者缺陷修复分支上的代码合并到发布分支上，以保证发布上线的代码是稳定的、干净的和可靠的。代码分支贯穿于整个软件开发的开发周期中，开发团队的成员会创建较多的代码分支以实现不同的功能。为维护好代码处理项目中较多的代码分支，通常开发团队会选定一种标准工作流，按照标准工作流的分支策略来进行代码分支的维护。

[0053] 图1是本申请实施例提供的一种实施环境的示意图。参见图1，该实施环境中包括终端101和服务器102。

[0054] 终端101可以为电脑、手机、平板电脑或者其他电子设备。服务器102可以是一台服务器，或者由若干台服务器组成的服务器集群，或者是一个云计算服务中心。

[0055] 终端101和服务器102之间通过无线或者有线网络连接。并且，终端101上可以安装有服务器102提供服务的客户端，终端101对应的用户可以通过该客户端实现例如数据传输、消息交互等功能。终端101对应的用户可以为任一代码处理项目的任一用户，该用户可以为该代码处理项目对应的开发团队的成员。该用户可以通过终端101上的客户端实现代码的开发、代码分支的创建、代码的提交、代码分支的合并和代码分支的销毁、代码的发布上线等代码处理操作。例如，该客户端可以为Git、Git GUI(Graphical User Interface，图形用户界面)工具或者IDE(Integrated Development Environment，集成开发环境)集成的Git客户端等。

[0056] 图2是本申请实施例提供的一种代码处理方法的示意图。本申请实施例提供的代码处理方法主要包括工作流识别与监测服务和工作流监测报告服务。工作流识别是在软件开发项目的初期或者在软件开发项目接入本申请实施例提供的代码处理方法时，识别该软件开发项目对应的标准工作流类型，即识别该软件项目所属的是哪一种标准工作流。工作流识别主要依据标准工作流的代码分支的特点来进行识别。

[0057] 识别出该软件开发项目所属的标准工作流类型之后，在该软件开发项目后续的整个软件开发周期中，持续的监测该软件开发项目对应的开发团队的各个成员在代码分支上执行的代码提交情况和代码分支的变化情况等；将代码提交情况和代码分支的变化情况与识别出的标准工作流类型对应的操作规范进行对比；如果代码提交情况或者代码分支的变化情况与识别出的标准工作流类型对应的操作规范不匹配，记录与该操作规范不匹配的代码提交情况或者代码分支的变化情况；并且，提供相应的正确的操作规范，也即提供相应的解决方案；并且，将相应的监测结果提供给工作流监测报告服务。

[0058] 工作流监测报告服务主要依据监测结果，向开发团队或者开发团队的单个成员展示与操作规范不匹配的代码提交情况或者代码分支的变化情况、相应的解决方案以及按照解决方案解决后的效果。该工作流监测报告服务可以通过网站展示和邮件通知等多种方式实现。例如，该工作流监测报告服务可以通过网站向开发团队展示整个软件开发周期中涉及到的与操作规范不匹配的代码提交情况或者代码分支的变化情况、相应的解决方案以及

按照解决方案解决后的效果。再如，该工作流监测报告服务可以通过邮件通知与操作规范不匹配的代码提交情况或者代码分支的变化情况对应的成员。在监测到与操作规范不匹配的代码提交情况或者代码分支的变化情况时，及时向对应的成员发送提示邮件。该提示邮件用于提示该成员操作不规范。该提示邮件中可以包括该成员执行的与操作规范不匹配的代码提交情况或者代码分支的变化情况和相应的解决方案。

[0059] 工作流识别与监测服务和工作流监测报告服务是相互依赖的，工作流监测报告服务就是展示工作流识别与监测服务的监测结果。

[0060] 图3是本申请实施例提供的一种代码处理方法的流程图。参见图3，该实施例包括：

[0061] 301、服务器基于任一代码处理项目的至少一个代码分支，以及在至少一个代码分支上所执行的至少一个第一代码处理操作，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型，第一工作流类型对应有至少一个操作规范。

[0062] 代码处理项目是为了更好的管理软件开发过程中的代码而创建的。例如，该代码处理项目可以为Git项目。通常，一个软件开发项目对应一个代码处理项目。开发团队的成员基于其所属的代码处理项目，执行例如代码的提交、代码分支的创建与合并、代码的销毁和代码的发布上线等代码处理操作。

[0063] 标准工作流提供了标准的代码分支维护方案。该代码分支维护方案包括该标准工作流对应的分支策略。对于任一代码分支，该分支策略中可以包括代码分支的创建策略、代码分支的销毁策略、代码分支的功能策略、代码分支的合并策略等。

[0064] 例如，代码分支的创建策略包括代码分支创建的时机，即何时创建该代码分支；代码分支的创建策略包括代码分支创建的位置，即该代码分支应基于哪个代码分支创建。代码分支的销毁策略包括代码分支销毁的时机，即何时销毁该代码分支。代码分支的功能策略包括代码分支对应的功能，例如，某些代码分支用于进行软件的功能开发、某些代码分支用于进行缺陷的修复、某些代码分支用于进行发布上线。代码分支的合并策略包括代码分支合并的方向，例如，第三代码分支应当合并到第四代码分支。

[0065] 操作规范用于约束在代码分支上执行的代码处理操作。操作规范是基于标准工作流对应的分支策略产生的。

[0066] 需要说明的一点是，标准工作流类型可以包括GitHub(一个面向开源及私有软件项目的托管平台)工作流、Git Flow工作流(一种Git的使用规范)、GitLab(一个用于仓库管理系统的开源项目)工作流和主干开发工作流等四种类型的工作流。其中GitLab工作流包括GitLab工作流的环境分支方案和GitLab工作流的发布分支方案。标准工作流类型还可以包括当前提出的其他工作流类型，在本申请实施例中不做限定。

[0067] 服务器可以根据开发团队的成员在代码分支上执行的至少一个第一代码处理操作，识别出该开发团队对应的代码处理项目所属的标准工作流类型，也即识别出该开发团队对应的代码处理项目属于哪一种标准工作流。

[0068] 在一种可能的实现方式中，不同类型的标准工作流对应的代码分支的特点是不同的。服务器可以根据不同类型的标准工作流对应的代码分支的特点确定该开发团队对应的代码处理项目所属的标准工作流类型。相应的，服务器基于任一代码处理项目的至少一个代码分支，以及在至少一个代码分支上所执行的至少一个第一代码处理操作，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型的步骤可以为：

服务器根据默认分支、发布分支、默认分支上的代码提交的频繁度和永久分支的数量中的至少一个，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型。

[0069] 从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型可以通过以下步骤(1)至(3)实现：

[0070] (1)服务器确定至少一个第一代码处理操作对应的第一工作流类型是否为主干分支工作流。

[0071] 在代码处理项目中，首次执行的代码处理操作对应的代码分支为默认分支。例如，默认分支可以为master(最重要的)分支。发布分支是用于进行代码的发布的代码分支。永久分支为在整个代码处理项目中持续存在的代码分支。

[0072] 标准工作流主要包括GitHub工作流、Git Flow工作流、主干开发工作流、GitLab环境分支工作流和GitLab发布分支工作流。按照默认分支上的代码提交的频繁度可以将上述五种类型的标准工作流分为两大类。参见表1，标准工作流可以分为主干开发类和特性分支开发类。

[0073] 表1

[0074]

分类	工作流
主干开发类	主干开发工作流
特性分支开发类	GitHub 工作流、Git Flow 工作流、GitLab 环境分支工作流、GitLab 发布分支工作流

[0075] 主干开发类的开发是在默认分支上进行的。主干开发类的默认分支上会进行频繁的代码提交。主干开发类包括主干开发工作流(Trunk Based development，TBD)。特性分支开发类的开发是在特性分支上开发的，特性分支开发类的默认分支上不会进行频繁的代码提交，只有必要时才会将在特性分支上开发的代码合并到默认分支上，因此，默认分支上的代码很少变动。

[0076] 需要说明的一点是，特性分支是一种短期分支，用于实现单一特性或其相关工作。例如，特性分支可以为hotfix(修补程序)分支。

[0077] 服务器基于默认分支上代码提交的频繁度，可以确定至少一个第一代码处理操作对应的第一工作流类型是否为主干开发工作流。相应的，根据默认分支、发布分支、默认分支上的代码提交的频繁度和永久分支的数量中的至少一个，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型的步骤可以为：当默认分支上的代码提交的频繁度大于预设阈值时，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为主干开发工作流。

[0078] 预设阈值可以为不超过1的数值。例如，预设阈值可以为0.8、0.9或者0.95等。

[0079] 默认分支上代码提交的频繁度可以根据代码提交操作对应的代码分支确定。相应的，当第一代码处理操作为代码提交操作时，服务器根据代码提交操作，确定默认分支上的代码提交的频繁度的步骤可以为：当第一代码处理操作为代码提交操作时，服务器根据代码提交操作，确定该代码提交操作对应的代码分支；将当前在默认分支上执行代码提交的代码提交操作的次数与当前执行的代码提交操作的总次数的比值作为默认分支上代码提

交的频繁度。

[0080] 需要说明的另一点是,服务器在确定默认分支上的代码提交的频繁度之前,可以通过以下命令确定默认分支:

[0081] git symbolic-ref refs/remotes/origin/HEAD|sed's@^refs/remotes/origin/@@'

[0082] (2)服务器确定至少一个第一代码处理操作对应的第一工作流类型是否为GitLab环境分支工作流或者Git Flow工作流。

[0083] 如果服务器根据默认分支上代码提交的频繁度,确定至少一个第一代码处理操作对应的第一工作流类型不是主干开发工作流,服务器还可以根据GitHub工作流、Git Flow工作流、GitLab环境分支工作流和GitLab发布分支工作流在永久分支的数量、发布分支等方面的差异,确定至少一个第一代码处理操作对应的第一工作流类型为特性分支开发类中的哪一种工作流类型。

[0084] Git Flow工作流有两个永久分支,默认分支和develop(开发)分支。GitHub工作流只有一个永久分支,该永久分支为默认分支。GitLab环境分支工作流有三个永久分支,分别为默认分支、pre-production(预发布)分支和production(发布)分支。GitLab发布分支工作流只有一个永久分支,该永久分支为默认分支。GitLab发布分支工作流在代码需要发布时会从默认分支创建release(发布)分支进行发布。

[0085] 服务器根据永久分支的数量,至少能够确定至少一个第一代码处理操作对应的第一工作流类型是否为GitLab环境分支工作流或者Git Flow工作流。相应的,服务器确定至少一个第一代码处理操作对应的第一工作流类型是否为GitLab环境分支工作流或者Git Flow工作流的步骤可以为:当永久分支的数量为第一数值时,服务器从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为GitLab环境分支工作流;当永久分支的数量为第二数值时,服务器从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为Git Flow工作流。其中,第一数值可以为3,第二数值可以为2。

[0086] 服务器根据永久分支的数量,至少能够确定至少一个第一代码处理操作对应的第一工作流类型是否为GitLab环境分支工作流或者Git Flow工作流之前,还需要确定永久分支的数量。相应的,服务器从代码分支中确定永久分支的数量的步骤可以为:对于代码处理项目中的每个代码分支,服务器获取每个代码分支的创建时间和销毁时间;根据每个代码分支的创建时间和销毁时间,从该代码处理项目的至少一个代码分支中确定永久分支。

[0087] 永久分支自代码处理项目创建开始就一直存在。服务器根据每个代码分支的创建时间和销毁时间,从该代码处理项目的至少一个代码分支中确定永久分支的步骤可以为:服务器获取代码处理项目的创建时间,如果第五代码分支的创建时间与代码处理项目的创建时间相同或者第五代码分支的创建时间和代码处理项目的创建时间之间没有创建其他代码分支,并且,第五代码分支从第五代码分支的创建时间至当前时间未被销毁,确定该第五代码分支为永久分支。

[0088] (3)服务器确定至少一个第一代码处理操作对应的第一工作流类型是否为GitHub工作流或者GitLab发布分支工作流。

[0089] GitHub工作流是在默认分支上进行代码的发布上线的,GitHub工作流对应的默认

分支和发布分支是同一个代码分支。GitLab发布分支工作流是在release分支上进行代码的发布上线的，release分支是基于默认分支创建的，GitLab发布分支工作流的默认分支和发布分支为不同的代码分支。

[0090] GitHub工作流和GitLab发布分支工作流永久分支的数量均为1，服务器可以根据默认分支和发布分支是否为同一个代码分支，确定至少一个第一代码处理操作对应的第一工作流类型为GitHub工作流或者GitLab发布分支工作流中的哪一个。相应的，服务器确定至少一个第一代码处理操作对应的第一工作流类型是否为GitHub工作流或者GitLab发布分支工作流的步骤可以为：当永久分支的数量为第三数值，且发布分支和默认分支为同一个代码分支时，服务器从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为GitHub工作流；当永久分支的数量为第三数值，且发布分支和默认分支为不同的代码分支时，服务器从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为GitLab发布分支工作流。其中，第三数值可以为1。

[0091] 服务器根据默认分支和发布分支是否为同一个代码分支，确定至少一个第一代码处理操作对应的第一工作流类型是否为GitHub工作流或者GitLab发布分支工作流之前，还需要确定从代码处理项目对应的代码分支中的发布分支。相应的，服务器确定发布分支的步骤可以为：当第一代码处理操作为代码发布操作时，服务器从代码分支中确定代码发布操作对应的发布分支，代码发布操作用于在发布分支上进行代码发布。

[0092] 在本申请实施例中，服务器能够根据默认分支、发布分支、默认分支上的代码提交的频繁度和永久分支的数量中的至少一个，自动识别出第一代码处理操作对应的第一工作流类型，识别效率较高。

[0093] 在另一种可能的实现方式中，服务器还可以结合上述五种类型的标准工作流在代码分支的分支线路图和永久分支上的代码活跃度等方面的差异，确定至少一个第一代码处理操作对应的第一工作流类型为标准工作流类型中的哪一个。代码分支的分支线路图包括代码分支创建的方向和代码分支合并的方向等信息。

[0094] 表2是特性分支开发类中的GitHub工作流、Git Flow工作流、GitLab环境分支工作流和GitLab发布分支工作流在永久分支的数量、分支线路图、永久分支上的代码活跃度和发布分支等四个方面的区别对比表，参见表2。

[0095] 表2

[0096]

标准工作流	永久分支的数量	分支线路图	代码活跃度	发布分支
Git Flow 工作流	2	从默认分支创建develop分支	默认分支上极少进行代码提交，develop分支的代码提交相对默认分支较多一些	默认分支
GitHub 工作流	1	从默认分支创建特性分支进行开发，将特性分支合并回默认分支进行发布	默认分支上的代码提交活动很频繁	默认分支
GitLab 环境分支工作流	3	默认分支合并到pre-production分支，pre-production分支合并到production分支	production分支上的代码提交活动最不频繁，其次是pre-production分支，最后是默认分支	production分支
GitLab 发布分支工作流	1	从默认分支创建release分支进行发布	默认分支上的代码提交活动相对较频繁	release分支

[0097] 在分支线路图方面,Git Flow工作流会从默认分支创建另一个永久分支develop分支。GitHub工作流只有一个默认分支,GitHub工作流会从默认分支上创建特性分支,在特性分支上进行开发;在特性分支上完成开发后,会将特性分支上的代码合并到默认分支上;在默认分支上进行代码的发布。GitLab环境分支工作流的代码分支的合并遵从“上游优先原则”。“上游优先原则”即代码只会从上游分支合并到下游分支,而不会从下游分支合并到上游分支。图4是本申请实施例提供的一种“上游优先原则”的示意图,参见图4,对于GitLab环境分支工作流,默认分支是pre-production分支的上游分支,pre-production分支是production分支的上游分支。“上游优先原则”也即代码只能从默认分支合并到pre-production分支,再从pre-production分支合并到production分支。对于GitLab环境分支工作流,代码不会从pre-production分支反向合并到默认分支,也不会从production分支反向合并到pre-production分支,也不会从默认分支跳跃合并到production分支。GitLab发布分支工作流只有在需要发布稳定版本的代码时,才会从默认分支创建release分支,在

release分支上进行代码的发布。GitLab发布分支工作流同样也遵从“上游优先原则”。对于GitLab发布分支工作流，默认分支是release分支的上游分支，代码只能从默认分支合并到release分支，而不会从release分支合并到默认分支。

[0098] 不同的标准工作流在永久分支上的代码活跃度是不同的，也即不同标准工作流在永久分支上的代码提交的频繁度是不同的。在永久分支上的代码活跃度方面，Git Flow工作流的默认分支极少进行代码提交，只有在有新的版本的代码需要发布的时候，才会将其他代码分支的代码合并到默认分支上。在用于进行功能开发的代码分支和用于修正缺陷的bugfix(修正缺陷)分支上开发完成后，都会将开发完成的代码合并到develop分支，因此，Git Flow工作流的develop分支上的代码提交相对于Git Flow工作流的默认分支较多一些。GitHub工作流只有一个默认分支，会从默认分支创建新的代码分支进行开发，因此，GitHub工作流的默认分支上的代码提交活动很频繁。对于GitLab环境分支工作流的默认分支、pre-production分支和production分支这三个永久分支，GitLab环境分支工作流的代码分支的合并遵从“上游优先原则”，会在默认分支上提交测试修改的代码，在默认分支上充分测试没有问题后才会将默认分支上的代码合并到pre-production分支；同样，在pre-production分支上提交测试修改的代码，在pre-production上充分测试没有问题后才会将默认分支上的代码合并到production分支。因此，对于GitLab环境分支工作流的默认分支、pre-production分支和production分支这三个永久分支，production分支上的代码提交活动最不频繁，其次，pre-production分支上的代码提交活动较不频繁，最后，默认分支上的代码提交活动比较频繁。

[0099] 不同的标准工作流用来进行代码发布的代码分支是不同的。在发布分支方面，Git Flow工作流在默认分支上进行代码的发布上线；GitHub工作流在默认分支上进行代码的发布上线；GitLab环境分支工作流在最下游的production分支上进行代码的发布上线；GitLab发布分支工作流从默认分支上创建的release分支，在release分支上进行代码的发布上线。

[0100] 图5是本申请实施例提供的一种确定第一工作流类型的流程图，参见图5，服务器还可以结合上述五种类型的标准工作流在代码分支的分支线路图和永久分支上的代码活跃度等方面的差异，确定至少一个第一代码处理操作对应的第一工作流类型为标准工作流类型中的哪一个。对于任一代码处理项目的至少一个代码分支，服务器根据在代码分支上执行的至少一个第一代码处理操作，确定是否在默认分支上频繁提交代码；如果在默认分支上频繁提交代码，且永久分支的数量为1，且永久分支与默认分支为同一个代码分支，则确定第一工作流类型为主干开发工作流；如果没有在主干分支上频繁提交代码，则根据永久分支的数量进一步确定第一工作流类型。相应的，如果永久分支的数量为3，且代码由默认分支合并到pre-production分支，由pre-production分支合并到production分支，则确定第一工作流类型为GitLab环境分支工作流；如果永久分支的数量为2，且从默认分支创建develop分支，则确定第一工作流类型为Git Flow工作流；如果永久分支的数量为1，且在默认分支上进行代码的发布，则确定第一工作流类型为GitHub工作流；如果永久分支的数量为1，且从默认分支上创建release分支，在release分支上进行代码的发布，确定第一工作流类型为GitLab发布分支工作流。

[0101] 在本申请实施例中，服务器还可以结合标准工作流在代码分支的分支线路图和永

久分支上的代码活跃度等方面的差异识别出第一代码处理操作对应的第一工作流类型，识别的结果更加准确。

[0102] 在另一种可能的实现方式中，开发团队在创建代码处理项目时，可以配置该代码处理项目对应的第一工作流类型为标准工作流类型中的哪一种。服务器可以根据开发团队的成员执行的第一代码处理操作，确定该成员执行的第一代码处理操作对应的代码处理项目，从而确定该代码处理项目对应的第一工作流类型。相应的，服务器基于任一代码处理项目的至少一个代码分支，以及在至少一个代码分支上所执行的至少一个第一代码处理操作，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型，包括：服务器根据第一代码处理操作，确定第一代码处理操作对应的代码处理项目；服务器确定该代码处理项目对应的配置信息，配置信息包括该代码处理项目对应的至少一个标准工作流类型中的第二工作流类型；服务器将第二工作流类型确定为至少一个第一代码处理操作对应的第一工作流类型。

[0103] 服务器根据第一代码处理操作，确定第一代码处理操作对应的代码处理项目，即确定该第一代码处理操作是基于哪一个代码处理项目执行的。例如，开发团队创建了第一代码处理项目，为该第一代码处理项目配置了配置信息，该配置信息用于指示该开发团队在该第一代码处理项目中应用GitHub工作流进行代码管理。该开发团队的成员A基于该第一代码处理项目执行了第一代码处理操作，该第一代码处理操作用于在默认分支上创建feature(特性)分支。服务器根据成员A执行的第一代码处理操作，确定该第一代码处理操作对应的代码处理项目为第一代码处理项目；服务器根据该第一代码处理项目对应的配置信息，确定该第一代码处理项目对应的标准工作流类型为GitHub工作流，即该至少一个第一代码处理操作对应的第一工作流类型为GitHub工作流。

[0104] 在本申请实施例中，开发团队可以配置其对应的代码处理项目对应的标准工作流类型，服务器可以直接根据配置信息确定第一代码处理操作对应的第一工作流类型，不需要根据标准工作流在默认分支、发布分支、默认分支上的代码提交的频繁度和永久分支的数量等方面的特点识别出第一代码处理操作对应的第一工作流类型，简单快捷，识别的速度较快。

[0105] 需要说明的一点是，除上述GitHub工作流、Git Flow工作流、主干开发工作流、GitLab环境分支工作流和GitLab发布分支工作流服务器等五种标准工作流外，服务器还可以通过机器学习的方法对代码处理的方式进行学习，也即对应用工作流的方式进行学习，将学习到的代码处理的方式作为标准工作流类型的一种。相应的，服务器通过机器学习的方法对代码处理的方式进行学习的步骤可以为：服务器基于任一代码处理项目的至少一个代码分支，以及在至少一个代码分支上所执行的至少一个第三代码处理操作，将至少一个第三代码处理操作输入到机器学习模型中，输出第三工作流类型，第三工作流类型对应有至少一个操作规范；将第三工作流类型确定为至少一个标准工作流类型中的一个。

[0106] 任一代码处理项目可以为服务器需要学习的代码处理项目。服务器基于需要学习的任一代码处理项目的至少一个代码分支，通过机器学习的方法学习在代码分支上所执行的至少一个第三代码处理操作，也即学习该代码处理项目进行代码处理的方式。服务器可以将学习到的代码处理的方式作为第三工作流类型，该第三工作流类型对应有至少一个操作规范，该至少一个操作规范可以用于监测应用该第三工作流类型的开发团队。

[0107] 例如,该任一代码处理项目可以为高效团队的代码处理项目。再如,该任一代码处理项目还可以为需要应用自定义的工作流的开发团队对应的代码处理项目。

[0108] 在本申请实施例中,除上述五种标准工作流类型外,服务器还能够通过机器学习的方法学习高效团队的代码处理方式,将学习到的高效团队对应的第三工作流类型作为标准工作流类型的一种,以第三工作流类型为标准去监测其他开发团队。扩展了用于作为监测标准的标准工作流的种类,从而能够提供更全面的监测服务。

[0109] 在本申请实施例中,服务器还能够学习开发团队自定义的工作流,将学习到的第三工作流类型作为标准工作流类型的一种,该第三工作流类型可以作为自定义的工作流对应的开发团队的监测标准,从而能够为自定义的工作流对应的开发团队提供监测服务,扩展了工作流监测服务的服务范围。

[0110] 302、对于代码处理项目的任一代码分支,服务器获取在代码分支上进行的任一第二代码处理操作,从第一工作流类型对应的至少一个操作规范中确定第二代码处理操作对应的第一目标操作规范,将第二代码处理操作与第一目标操作规范进行比对。

[0111] 服务器从至少一个标准工作流类型中确定第一代码处理操作对应的第一工作流类型后,基于该第一工作流类型对应的操作规范监测开发团队的成员是否正确执行了代码处理操作。

[0112] 在一种可能的实现方式中,服务器可以从历史日志记录中获取第二代码处理操作,将该第二代码处理操作与该第二代码处理操作对应的第一目标操作规范进行对比。相应的,对于代码处理项目的任一代码分支,服务器获取在代码分支上进行的任一第二代码处理操作,从第一工作流类型对应的至少一个操作规范中确定第二代码处理操作对应的第一目标操作规范,将第二代码处理操作与第一目标操作规范进行比对的步骤可以为:服务器获取历史日志记录,历史日志记录用于记录在代码分支上执行的第二代码处理操作;根据历史日志记录中的第二代码处理操作;从第一工作流类型对应的至少一个操作规范中确定第二代码处理操作对应的第一目标操作规范,将第二代码处理操作与第一目标操作规范进行比对。

[0113] 例如,第一工作流类型为Git Flow工作流,第一工作流类型对应的操作规范包括只有一个默认分支、只有一个基于默认分支创建的develop分支,feature分支只能基于develop分支创建、feature分支只能合并到develop分支和feature分支合并到develop分支后,feature应被删除等。如果第二代码处理操作为基于默认分支创建feature分支,则第二代码处理操作对应的第一目标操作规范为feature分支只能基于develop分支创建。将第二代码处理操作与第一目标操作规范进行比对,第二代码处理操作与第一目标操作规范不匹配。

[0114] 在本申请实施例中,服务器能够根据开发团队的成员基于代码分支执行的代码处理操作,从第一工作流类型对应的操作规范中确定该代码处理操作对应的第一目标操作规范,将该代码处理操作与第一目标操作规范进行对比,实现对开发团队中的各个成员的代码处理操作的自动监测,自动监测出不正确的代码处理操作,监测效率较高。

[0115] 在另一种可能的实现方式中,服务器还可以将第二代码处理操作对应的第一代码分支上的分支变化与第一目标操作规范所确定的分支变化信息进行比对。相应的,服务器将第二代码处理操作与第一目标操作规范进行比对的步骤还可以为:服务器根据第二代码

处理操作，确定第二代码处理操作对应的第一代码分支；服务器根据第一代码分支上进行第二代码处理操作前后的代码，确定第一代码分支的分支变化；服务器将第一代码分支的分支变化与第一目标操作规范所确定的分支变化信息进行比对。

[0116] 例如，第一工作流类型为Git Flow工作流，第一工作流类型对应的操作规范包括只有一个默认分支、只有一个基于默认分支创建的develop分支，feature分支只能基于develop分支创建、feature分支只能合并到develop分支和feature分支合并到develop分支后，feature应被删除等。如果第二代码处理操作为将feature分支合并到develop分支，则第二代码处理操作对应的第一代码分支为develop分支，第二代码处理操作对应的第一目标操作规范为feature分支只能合并到develop分支和feature分支合并到develop分支后，feature应被删除，第一目标操作规范所确定的分支变化信息为develop分支上增加了feature分支对应的代码，feature分支被删除。服务器可以获取执行第二代码处理操作前后develop分支的分支变化，例如，develop分支的分支变化可以为develop分支上增加了feature分支对应的代码，且develop分支上仍存在feature分支。将该develop分支的分支变化与第一目标操作规范所确定的分支变化信息进行比对，该develop分支的分支变化与第一目标操作规范所确定的分支变化信息不匹配。

[0117] 在本申请实施例中，服务器能够根据代码处理操作对应的代码分支执行代码处理操作前后的分支变化和第一目标操作规范所确定的分支变化信息进行比对，自动监测代码处理操作是否准确的实现了应当实现的效果，能够使监测结果更加准确。

[0118] 需要说明的一点是，服务器能够基于上述GitHub工作流、Git Flow工作流、主干开发工作流、GitLab环境分支工作流和GitLab发布分支工作流等五种标准工作流类型的分支策略分别建立五种监测模型，每种监测模型中包括相应的标准工作流类型对应的操作规范。服务器可以从五种不同的监测模型中获取第一工作流类型对应的至少一个操作规范。例如，当第一工作流类型为GitHub工作流时，服务器可以从GitHub工作流对应的监测模型中获取至少一个操作规范。

[0119] Git Flow工作流的分支模型是最复杂的，涉及到的代码分支也是五种类型的标准工作流中最多的，Git Flow工作流由Vincent Driessen(文森特·德里森)于2010年提出，图6是本申请实施例提供的一种Git Flow工作流的分支模型的示意图，参见图6，Git Flow工作流对应有五种代码分支的类型：master(默认)分支、develop(开发)分支、feature branches(特性分支)、release branches(发布分支)、hotfixes(修补程序分支)。feature branches也即feature分支，releasebranches也即release分支，hotfixes也即hotfix分支。

[0120] 从master分支创建develop分支，在master分支上打tag(版本标签)0.1。如果版本0.1中包括severe bug(严重缺陷)需要修复，从master分支创建hotfix分支0.2，在hotfix分支0.2上修复缺陷后，将hotfix分支0.2同时合并到develop分支和master分支，在master分支上打tag0.2，删除hotfix分支0.2。

[0121] 从develop分支创建feature分支，对于下一次要分布的主要功能，在feature分支上进行开发，功能开发完成后，将相应的feature分支合并到develop分支。develop分支上可以同时创建多个feature分支，同时进行功能开发。

[0122] 当版本1.0准备发布时，从develop分支为版本1.0创建release分支，在release分

支上进行发布前的测试或问题解决等发布前检查，release分支上仅有bugfixes(修正缺陷)。如果在release分支上检查没有问题时，将release分支同时合并到master分支和develop分支，在master分支上打tag1.0，删除该release分支。

[0123] release分支上的bugfixes持续的合并回develop分支，再从develop分支创建feature分支，在feature分支上进行bugfixes的修复，修复完成后，将相应的feature分支合并到develop分支。再从develop分支为创建另一个版本对应的release分支，执行在release分支上进行发布前的测试或问题解决等发布前检查的步骤。

[0124] Git Flow工作流对应的分支策略以及对应的操作规范包括(1)至(17)：

[0125] (1)Git Flow工作流对应有五种代码分支的类型：master分支、develop分支、feature分支、release分支、hotfix分支。

[0126] (2)master分支和develop分支为主要分支，也即永久分支。feature分支、release分支和hotfix分支为支持分支，也即短暂分支。

[0127] (3)master分支和develop分支在一直存在于代码处理项目中，且master分支在一个代码处理项目中只有一个，develop分支在一个代码处理项目中只有一个。feature分支、release分支和hotfix分支在代码处理项目的生命周期中会不断的创建、删除。在代码处理项目的生命周期中分别有多个不同的feature分支，在代码处理项目的生命周期中分别有多个不同的release分支，在代码处理项目的生命周期中分别有多个不同的hotfix分支。

[0128] (4)develop分支应基于master分支创建，不能基于其他代码分支创建，并且develop分支只在代码处理项目的初始阶段创建一次，后续不需要再创建develop分支。

[0129] (5)feature分支应基于develop分支创建，不能基于其他代码分支创建。feature分支用于进行功能开发。例如，feature分支用于开发一个新的功能。

[0130] (6)在feature分支上的功能开发完成后，应将feature分支合并到develop分支，不能合并到其他代码分支。

[0131] (7)feature分支合并到develop分支后，应删除该feature分支，不能一直保留该feature分支。

[0132] (8)上述(5)至(7)会在代码处理项目的生命周期中反复发生，但同一个feature分支不会长期存在于代码处理项目的生命周期中，在代码处理项目的生命周期中会不断地创建新的feature分支，在该feature分支进行功能开发，将该feature分支合并到develop分支，删除该feature分支，再创建另一个新的feature分支。

[0133] (9)在代码准备发布时，应从develop分支创建release分支，用于发布前的测试、问题解决等发布前检查。release分支只能从develop分支创建，不能从其他代码分支创建。

[0134] (10)在release分支上检查没问题时，将release分支同时合并到master分支和develop分支，不能只将release分支合并到master分支。

[0135] (11)release分支合并到master分支和develop分支后，删除该release分支，不能一直保留该release分支。

[0136] (12)应从master分支进行代码的发布上线，不能从其他代码分支发布。

[0137] (13)在master分支进行代码的发布上线后，应对master分支打tag(标签)，只能在master分支上打tag，其他代码分支不需要打tag。

[0138] (14)在代码发布后或者发布上线的代码有bug(缺陷)时，从master分支创建

hotfix分支解决bug,不能从其他代码分支创建hotfix分支。

[0139] (15)在hotfix分支上解决bug后,将hotfix分支同时合并到master分支和develop分支,不能只将hotfix分支合并到master分支。

[0140] (16)hotfix分支合并到master分支和develop分支后,删除该hotfix分支,不能一直保留该hotfix分支。

[0141] (17)上述(14)至(16)会在代码处理项目的生命周期中反复发生,但同一个hotfix分支不会长期存在于代码处理项目的生命周期中。在代码处理项目的生命周期中,当需要解决bug时,就会创建新的hotfix分支,在该hotfix分支上解决bug,将该hotfix分支合并到master分支和develop分支,删除该hotfix分支,当再次需要解决bug时,再创建另一个新的hotfix分支。

[0142] 需要说明的一点是,上述(1)至(17)仅为Git Flow工作流对应的部分的分支策略和操作规范,Git Flow工作流还对应有其他分支策略和操作规范,在本申请实施例中不再一一描述。

[0143] GitHub工作流适用于持续集成的场景,GitHub工作流由GitHub于2011年提出,GitHub工作流最大的特点是代码分支简单,只有一个master分支为长期分支,也即永久分支。GitHub工作流比Git Flow工作流简单很多。图7是本申请实施例提供的一种GitHub工作流的分支模型的示意图,参见图7,GitHub工作流只有一个永久分支为master分支,从master分支创建feature分支或hotfix分支,用于开发功能和解决问题。在feature分支上功能开发完成或者在hotfix分支上解决完问题后,将feature分支或者hotfix分支合并到master分支,删除feature分支或者hotfix分支,在master分支上进行代码的发布上线。

[0144] GitHub工作流对应的分支策略以及对应的操作规范包括(18)至(23):

[0145] (18)GitHub工作流只有一个master分支是永久分支,GitHub工作流不能有第二个永久分支,GitHub工作流对应的其他代码分支都是短暂分支。

[0146] (19)由于GitHub工作流只有master分支一个永久分支,所以通常会对master分支设为protected(保护),如果不对master分支设为protected保护,则master分支不够安全。

[0147] (20)GitHub工作流从master分支创建feature分支,用于进行功能开发。GitHub工作流从master分支创建hotfix分支,用于进行问题解决。

[0148] (21)GitHub工作流将feature分支或hotfix分支的代码合并到master分支时,必须创建pull request,否则不规范。

[0149] (22)GitHub工作流将feature分支的代码合并到master分支后需要删除feature分支,不能一直保留feature分支。GitHub工作流将hotfix分支的代码合并到master分支后需要删除hotfix分支,不能一直保留hotfix分支。

[0150] (23)GitHub工作流从master分支上进行代码的发布上线,不能从其他的代码分支进行代码的发布上线。

[0151] 需要说明的一点是,上述(18)至(23)仅为GitHub工作流对应的部分的分支策略和操作规范,GitHub工作流还对应有其他分支策略和操作规范,在本申请实施例中不再一一描述。

[0152] GitLab工作流由GitLab于2014年提出,GitLab工作流同时集合了Git Flow工作流和GitHub工作流的优点,同时满足持续发布和版本发布的开发流程。GitLab工作流使用

Merge Request(合并请求)进行code review(代码检查)和分支合并,并且提供了issue tracking(问题追踪)。GitLab工作流提供了两种解决方案,一种是适合不同环境部署的Environment Branches(环境分支)方案,该方案对应GitLab环境分支工作流。另一种是适合版本发布的Release Branches(发布分支)方案,该方案对应GitLab发布分支工作流。

[0153] 图8是本申请实施例提供的一种GitLab环境分支工作流的分支模型的示意图,参见图8,GitLab环境分支工作流通常对应有开发环境、预发布环境和发布环境三个环境,每个环境对应一个分支。开发环境对应master分支,预发布环境对应pre-production分支,发布环境对应production分支。

[0154] GitLab环境分支工作流对应的分支策略以及对应的操作规范包括(24)至(29):

[0155] (24)GitLab环境分支工作流对应有master分支、pre-production分支和production分支三个永久分支。在一些特殊情况中,GitLab环境分支工作流可能只对应有master分支和production分支,不需要pre-production分支。

[0156] (25)GitLab环境分支工作流只能从master分支创建特性分支进行功能开发和bug解决,GitLab环境分支工作流不能从其他永久分支创建特性分支。

[0157] (26)功能开发完成后,应将特性分支对应的代码合并到master分支,不能将特性分支对应的代码直接合并到pre-production分支或production分支。

[0158] (27)在master分支上充分测试没有问题后,将master分支上的代码合并到pre-production分支,不能直接将master分支上的代码合并到production分支。

[0159] (28)在pre-production分支上充分测试没有问题后,将pre-production上的代码合并到production分支,不能将pre-production分支上的代码合并到master分支。

[0160] (29)从production分支上进行代码的发布上线,不能从其他代码分支进行代码的发布上线。

[0161] 需要说明的一点是,上述(24)至(29)仅为GitLab环境分支工作流对应的部分的分支策略和操作规范,GitLab环境分支工作流还对应有其他分支策略和操作规范,在本申请实施例中不再一一描述。

[0162] GitLab发布分支工作流主要用于对外发布软件的场景,在发布软件时,GitLab发布分支工作流主从master分支创建release分支,用于记录发布的版本。

[0163] 图9是本申请实施例提供的一种GitLab发布分支工作流的分支模型的示意图,参见图9,GitLab发布分支工作流在有稳定版本要发布时,从master分支创建发布分支,例如,发布分支2-3-stable(稳定的)、发布分支2-4-stable等。

[0164] GitLab发布分支工作流对应的分支策略以及对应的操作规范包括(30)至(38):

[0165] (30)GitLab发布分支工作流只有一个永久分支为master分支。

[0166] (31)GitLab发布分支工作流进行新功能开发的代码需要合并到master分支。

[0167] (32)GitLab发布分支工作流在有稳定版本要发布时,从master分支创建发布分支,每一个版本发布,都会创建一个发布分支。

[0168] (33)GitLab发布分支工作流从新创建的发布分支上进行软件发布,不能从master分支上直接发布。

[0169] (34)GitLab发布分支工作流发布后的发布分支保留,用于记录软件发布。

[0170] (35)软件发布后,当出现问题时,从出现问题的版本的发布分支创建修复分支,进

行问题修复。

[0171] (36)问题修复完成之后,将修复分支合并到master分支,不能将修复分支直接合并到发布分支。

[0172] (37)在master分支上充分测试没有问题之后,将master分支合并到相应的发布分支。

[0173] (38)在出现问题的版本的发布分支上发布问题修复后的版本。

[0174] 需要说明的一点是,上述(30)至(38)仅为GitLab发布分支工作流对应的部分的分支策略和操作规范,GitLab发布分支工作流还对应有其他分支策略和操作规范,在本申请实施例中不再一一描述。

[0175] 主干开发工作流与上述其他四种标准工作流有较大不同,主干开发工作流的开发是在主干分支,也即master分支上进行的上述其他四种标准工作流都是在特性分支上进行开发,不会直接在master分支上进行开发。在主干分支上进行开发可以避免分支合并的困扰,保证随时拥有可发布的版本,但由于所有开发团队的成员都在同一个分支工作,开发团队需要合理的分工和充分沟通才能保证不同成员的代码尽可能少的发生冲突。图10是本申请实施例提供的一种主干开发工作流的分支模型的示意图,参见图10,开发团队的开发人员在主干分支上提交代码。如果仅仅依靠打tag不能满足发布需求,则从主干分支上创建发布分支,例如,发布分支的版本可以为1.1.0、1.1.1、1.2.0等。开发团队的发布工程师可以将代码合并到发布分支,代码合并到发布分支后,在发布分支上打tag。发布工程师还可以在开发人员完成bug修复后,将修复完成的bug,cherry-pick(挑拣提交)到发布分支。

[0176] 主干开发工作流对应的分支策略以及对应的操作规范包括(39)至(44):

[0177] (39)主干开发工作流只有一个永久分支为master分支,所有新功能对应的代码的提交都提交到master分支上。

[0178] (40)对于主干开发工作流,原则上来说,服务器仓库中不能有其他代码分支,当有发布需要时,可以有发布分支。

[0179] (41)当需要进行代码的发布时,在master分支上打tag来标记发布的版本。

[0180] (42)如果仅仅依靠打tag不能满足发布需求,则从主干分支上创建发布分支。

[0181] (43)bug修复在主干分支上进行。

[0182] (44)bug修复完成之后,再将修复完成的bug,cherry-pick到发布分支。

[0183] 需要说明的一点是,上述(39)至(44)仅为主干开发工作流对应的部分的分支策略和操作规范,主干开发工作流还对应有其他分支策略和操作规范,在本申请实施例中不再一一描述。

[0184] 303、若第二代码处理操作与第一目标操作规范不匹配,服务器输出第二代码处理操作对应的监测结果,该监测结果至少包括第二代码处理操作对应的第一目标操作规范。

[0185] 服务器可以向第二代码处理操作对应的开发团队的成员输出监测结果,服务器将在整个代码处理项目中监测到的每一个与第一目标操作规范不匹配的第二代码处理操作对应的监测结果整合起来,向开发团队输出,以便开发团队在软件开发项目完成后进行整理和总结。

[0186] 在一种可能的实现方式中,服务器可以将监测结果输出给第二代码处理操作对应的开发团队的成员。相应的,若第二代码处理操作与第一目标操作规范不匹配,服务器输出

第二代码处理操作对应的监测结果的步骤可以为:服务器根据第二代码处理操作,确定该第二代码处理操作对应的用户信息;根据该用户信息对应的邮件地址;向该邮件地址输出监测结果,该监测结果至少包括该第二代码处理操作对应的第一目标操作规范。

[0187] 在本申请实施例中,服务器可以向第二代码处理操作对应的用户输出监测结果,该监测结果中至少包括该用户执行的第二代码处理操作对应的第一目标操作规范,能够提示该用户根据第一目标操作规范正确的执行相应的代码处理操作,从而能够使用户轻松高效的应用标准工作流,提高用户所属的开发团队的协作开发效率。

[0188] 在另一种可能的实现方式中,该监测结果中还可以包括该第二代码处理操作对应的操作信息和该用户执行该第二代码处理操作的时间等信息。

[0189] 在本申请实施例中,该监测结果中还包括第二代码处理操作对应的操作信息和该用户执行该第二代码处理操作的时间等信息,能够辅助用户快速定位到不符合第一目标操作规范的第二代码处理操作,提高用户应用标准工作流的效率。

[0190] 在另一种可能的实现方式中,服务器还可以将整个代码处理项目中的监测结果输出给开发团队。相应的,若第二代码处理操作与第一目标操作规范不匹配,服务器输出第二代码处理操作对应的监测结果的步骤可以为:在代码处理项目完成后,服务器获取该代码处理项目对应的至少一个监测结果,以及,对于至少一个监测结果中的每个监测结果,确定该监测结果对应的修正结果;根据该监测结果和该修正结果,生成监测报告;将该监测报告输出给开发团队。

[0191] 服务器向开发团队的成员输出监测结果后,还可以继续监测与第一目标操作规范不匹配的第二代码处理操作对应的第二代码分支的分支变化,如果确定与第一目标操作规范不匹配的第二代码处理操作已被修正,记录修正结果。该修正结果中包括与第一目标操作规范不匹配的第二代码处理操作被修正后对整个代码处理项目的有益效果。

[0192] 服务器可以获取该代码处理项目对应的所有监测结果,服务器也可以从该代码处理项目对应的所有监测结果获取具有代表性的至少一个监测结果,服务器还可以根据与第一目标操作规范不匹配的第二代码处理操作的类型对该代码处理项目对应的所有监测结果进行分类,从每一类监测结果中获取至少一个监测结果。在本申请实施例中,对此不做限定。

[0193] 在本申请实施例中,服务器能够针对整个代码处理项目对应的监测结果和修正结果,生成监测报告,输出给开发团队,为开发团队总结与整理标准工作流的应用情况提供便利,增强开发团队的成员对标准工作流的理解,提高后续团队协作开发的效率。

[0194] 在另一种可能的实现方式中,服务器输出监测结果的同时,还可以根据第一目标操作规范,自动修正与第一目标操作规范不匹配的第二代码处理操作。相应的,若第二代码处理操作与第一目标操作规范不匹配,服务器输出监测结果的步骤可以为:若第二代码处理操作与第一目标操作规范不匹配,服务器生成第二代码处理操作对应的监测结果,以及,若第二代码处理操作与第一目标操作规范不匹配,服务器根据与第一目标操作规范不匹配的第二代码处理操作和第一目标操作规范,对该第二代码处理操作对应的第二代码分支执行第四代码处理操作;输出监测结果。

[0195] 第一目标操作规范所对应的第四代码处理操作用于在第二代码分支上执行相应的代码处理操作,以修正与第一目标操作规范不匹配的第二代码处理操作在第二代码分支

上执行的不正确的操作。

[0196] 例如，第一工作流类型为Git Flow工作流；第二代码处理操作为将第一feature分支合并到develop分支，第一feature分支未删除；第二代码处理操作对应的第一目标操作规范为feature分支只能合并到develop分支和feature分支合并到develop分支后，feature分支应被删除。该第二代码处理操作与第一目标操作规范不匹配，为修正该与第一目标操作规范不匹配的第二代码处理操作应执行的第四代码处理操作为删除develop分支上的第一feature分支。

[0197] 在本申请实施例中，服务器不仅能够输出监测结果，还能根据第二代码处理操作和第一目标操作规范，自动修正与第一目标操作规范不匹配的第二代码处理操作，从而能够提高标准工作流应用的准确性，提高开发效率。

[0198] 需要说明的一点是，服务器从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型之后，还可以根据第一工作流类型对应的操作规范对已执行的第一代码处理操作进行检查。相应的，服务器对已执行的第一代码处理操作进行检查的步骤可以为：对于至少一个第一代码处理操作中的每一个第一代码处理操作，服务器从第一工作流类型对应的至少一个操作规范中确定第一代码处理操作对应的第二目标操作规范，将第一代码处理操作与第二目标操作规范进行比对；若第一代码处理操作与第二目标操作规范不匹配，根据第一代码处理操作与第二目标操作规范，确定第一代码处理操作对应的比对结果。

[0199] 例如，第一工作流类型为Git Flow工作流，第一代码处理操作为将release分支合并到默认分支，则该第一代码处理操作对应的第二目标操作规范为release分支应同时合并到默认分支和develop分支。将该第一代码处理操作与第二目标操作规范进行比对，该第一代码处理操作与第二目标操作规范不匹配，则该第一代码处理操作对应的比对结果为该第一代码处理操作应将release分支同时合并到默认分支和develop分支，但该第一代码处理操作仅将release分支合并到默认分支。

[0200] 服务器可以将比对结果输出给该第一代码处理操作对应的开发团队的成员，提示该成员其执行的第一代码处理操作与操作规范不符。服务器也可以不向开发团队的成员输出比对结果。服务器还可以将比对结果存储在缓存区中，当开发团队需要进行代码检查时，可以从缓存区中获取对比结果，通过对比结果来辅助代码检查。

[0201] 在本申请实施例中，服务器确定至少一个第一代码处理操作对应的第一工作流类型之后，还可以根据第一工作流类型对已执行的第一代码处理操作进行检查。从而服务器能够实现对整个代码处理项目中所有代码处理操作的监测，代码处理操作的监测更加全面。

[0202] 在本申请实施例中，对于任一代码处理项目的代码分支，服务器能够自动根据代码分支上所执行的第一代码处理操作，确定该代码处理项目对应的开发团队应用的第一工作流类型；监测代码分支上进行的第二代码处理操作，与第一工作流类型对应的第一目标操作规范进行比对；对于与第一目标操作规范不匹配的第二代码处理操作，输出相应的监测结果，监测结果中至少包括第一目标操作规范。服务器能够确定任一代码处理项目对应的第一工作流类型，基于该第一工作流类型对应的操作规范，监测代码分支上执行的代码处理操作；对于未按照操作规范执行的代码处理操作，服务器能够输出至少包括第一目标

操作规范的监测结果，从而使得开发团队的成员能够基于该监测结果对未按照操作规范执行的代码处理操作及时进行修正。在本申请中，服务器能够辅助开发团队更加准确、高效的应用标准工作流，使开发团队协作开发的效率更高。

[0203] 上述所有可选技术方案，可以采用任意结合形成本申请的可选实施例，在此不再一一赘述。

[0204] 图11是本申请实施例提供的一种代码处理装置的框图。参见图11，该装置包括：

[0205] 确定模块1101，被配置为基于任一代码处理项目的至少一个代码分支，以及在至少一个代码分支上所执行的至少一个第一代码处理操作，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型，第一工作流类型对应有至少一个操作规范；

[0206] 比对模块1102，被配置为对于代码处理项目的任一代码分支，获取在代码分支上进行的任一第二代码处理操作，从第一工作流类型对应的至少一个操作规范中确定第二代码处理操作对应的第一目标操作规范，将第二代码处理操作与第一目标操作规范进行比对；

[0207] 输出模块1103，被配置为若第二代码处理操作与第一目标操作规范不匹配，输出第二代码处理操作对应的监测结果，监测结果至少包括第一目标操作规范。

[0208] 在一种可能的实现方式中，比对模块1102，还被配置为根据第二代码处理操作，确定第二代码处理操作对应的第一代码分支；根据第一代码分支上进行第二代码处理操作前后的代码，确定第一代码分支的分支变化；将第一代码分支的分支变化与第一目标操作规范所确定的分支变化信息进行比对。

[0209] 在另一种可能的实现方式中，代码分支包括默认分支、永久分支和发布分支；确定模块1101，还被配置为根据默认分支、发布分支、默认分支上的代码提交的频繁度和永久分支的数量中的至少一个，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型。

[0210] 在另一种可能的实现方式中，标准工作流类型包括主干开发工作流、GitFlow工作流、GitHub工作流、GitLab环境分支工作流和GitLab发布分支工作流；确定模块1101，还被配置为当默认分支上的代码提交的频繁度大于预设阈值时，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为主干开发工作流；当永久分支的数量为第一数值时，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为GitLab环境分支工作流；当永久分支的数量为第二数值时，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为Git Flow工作流；当永久分支的数量为第三数值，且发布分支和默认分支为同一个代码分支时，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为GitHub工作流；当永久分支的数量为第三数值，且发布分支和默认分支为不同的代码分支时，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型为GitLab发布分支工作流。

[0211] 在另一种可能的实现方式中，确定模块1101，还被配置为当第一代码处理操作为代码发布操作时，从代码分支中确定代码发布操作对应的发布分支，代码发布操作用于在发布分支上进行代码发布；当第一代码处理操作为代码提交操作时，根据代码提交操作，确

定默认分支上的代码提交的频繁度;从代码分支中确定永久分支的数量。

[0212] 在另一种可能的实现方式中,确定模块1101,还被配置为根据第一代码处理操作,确定第一代码处理操作对应的代码处理项目;确定代码处理项目对应的配置信息,配置信息包括代码处理项目对应的至少一个标准工作流类型中的第二工作流类型;将第二工作流类型确定为至少一个第一代码处理操作对应的第一工作流类型。

[0213] 在另一种可能的实现方式中,该装置还包括:

[0214] 机器学习模块,被配置为基于任一代码处理项目的至少一个代码分支,以及在至少一个代码分支上所执行的至少一个第三代码处理操作,将至少一个第三代码处理操作输入到机器学习模型中,输出第三工作流类型,第三工作流类型对应有至少一个操作规范;

[0215] 确定模块1101,还被配置为将第三工作流类型确定为至少一个标准工作流类型中的一个。

[0216] 在另一种可能的实现方式中,输出模块1103,还被配置为若第二代码处理操作与第一目标操作规范不匹配,生成第二代码处理操作对应的监测结果,以及,若第二代码处理操作与第一目标操作规范不匹配,根据第二代码处理操作和第一目标操作规范,对第二代码处理操作对应的第二代码分支执行第四代码处理操作;输出监测结果。

[0217] 在另一种可能的实现方式中,该装置还包括:

[0218] 比对模块1102,还被配置为对于至少一个第一代码处理操作中的每一个第一代码处理操作,从第一工作流类型对应的至少一个操作规范中确定第一代码处理操作对应的第二目标操作规范,将第一代码处理操作与第二目标操作规范进行比对;

[0219] 确定模块1101,还被配置为若第一代码处理操作与第二目标操作规范不匹配,根据第一代码处理操作与第二目标操作规范,确定第一代码处理操作对应的比对结果。

[0220] 在本申请实施例中,对于任一代码处理项目的代码分支,服务器能够自动根据代码分支上所执行的第一代码处理操作,确定该代码处理项目对应的开发团队应用的第一工作流类型;监测代码分支上进行的第二代码处理操作,与第一工作流类型对应的第一目标操作规范进行比对;对于与第一目标操作规范不匹配的第二代码处理操作,输出相应的监测结果,监测结果中至少包括第一目标操作规范。服务器能够确定任一代码处理项目对应的第一工作流类型,基于该第一工作流类型对应的操作规范,监测代码分支上执行的代码处理操作;对于未按照操作规范执行的代码处理操作,服务器能够输出至少包括第一目标操作规范的监测结果,从而使得开发团队的成员能够基于该监测结果对未按照操作规范执行的代码处理操作及时进行修正。在本申请中,服务器能够辅助开发团队更加准确、高效的应用标准工作流,使开发团队协作开发的效率更高。

[0221] 需要说明的是:上述实施例提供的代码处理装置在代码处理时,仅以上述各功能模块的划分进行举例说明,实际应用中,可以根据需要而将上述功能分配由不同的功能模块完成,即将服务器的内部结构划分成不同的功能模块,以完成以上描述的全部或者部分功能。另外,上述实施例提供的代码处理装置与代码处理方法实施例属于同一构思,其具体实现过程详见方法实施例,这里不再赘述。

[0222] 图12是本申请实施例提供的一种服务器的框图,该服务器1200可因配置或性能不同而产生比较大的差异,可以包括一个或一个以上处理器(CentralProcessing Units,CPU)1201和一个或一个以上的存储器1202,其中,所述存储器1202中存储有至少一条程序

代码，所述至少一条程序代码由所述处理器1201加载并执行以实现上述各个方法实施例提供的方法。当然，该服务器还可以具有有线或无线网络接口、键盘以及输入输出接口等部件，以便进行输入输出，该服务器还可以包括其他用于实现设备功能的部件，在此不做赘述。

[0223] 在示例性实施例中，还提供了一种计算机可读存储介质，该计算机可读存储介质中存储有至少一条程序代码，上述至少一条程序代码可由服务器中的处理器执行以完成上述实施例中的代码处理方法。例如，所述计算机可读存储介质可以是ROM(Read-Only Memory，只读存储器)、RAM(随机存取存储器，Random Access Memory)、CD-ROM(Compact Disc Read-Only Memory，只读光盘)、磁带、软盘和光数据存储设备等。

[0224] 本申请还提供了一种计算机程序产品，所述计算机程序产品包括一个或多个计算机程序，所述计算机程序被处理器执行时，用于实现上述各个方法实施例提供的代码处理方法。

[0225] 本领域普通技术人员可以理解实现上述实施例的全部或部分步骤可以通过硬件来完成，也可以通过程序来指令相关的硬件完成，所述的程序可以存储于一种计算机可读存储介质中，上述提到的存储介质可以是只读存储器，磁盘或光盘等。

[0226] 以上所述仅为本申请的可选实施例，并不用以限制本申请，凡在本申请的精神和原则之内，所作的任何修改、等同替换、改进等，均应包含在本申请的保护范围之内。

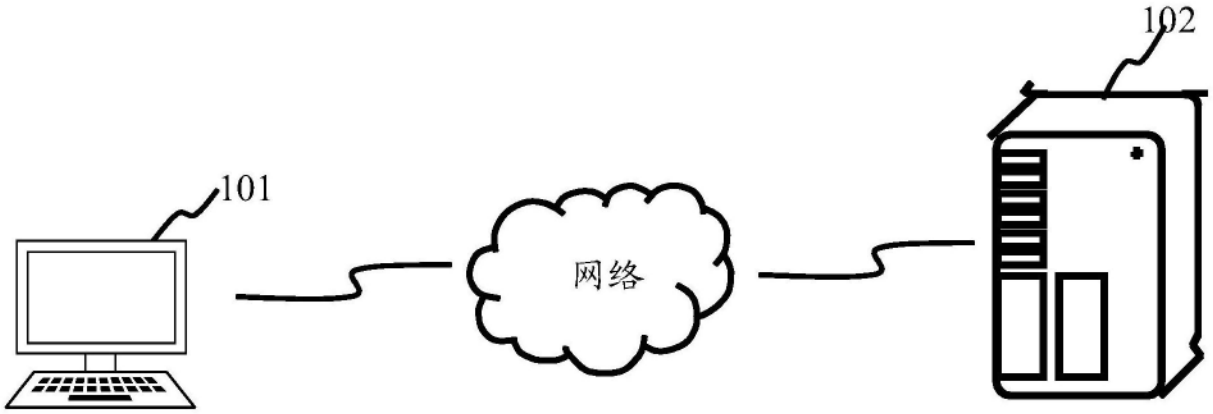

102
101
网络

图1

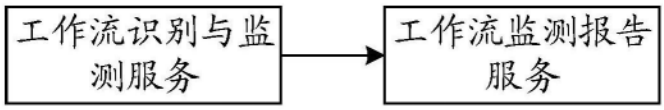

工作流识别与监测服务
工作流监测报告服务

图2

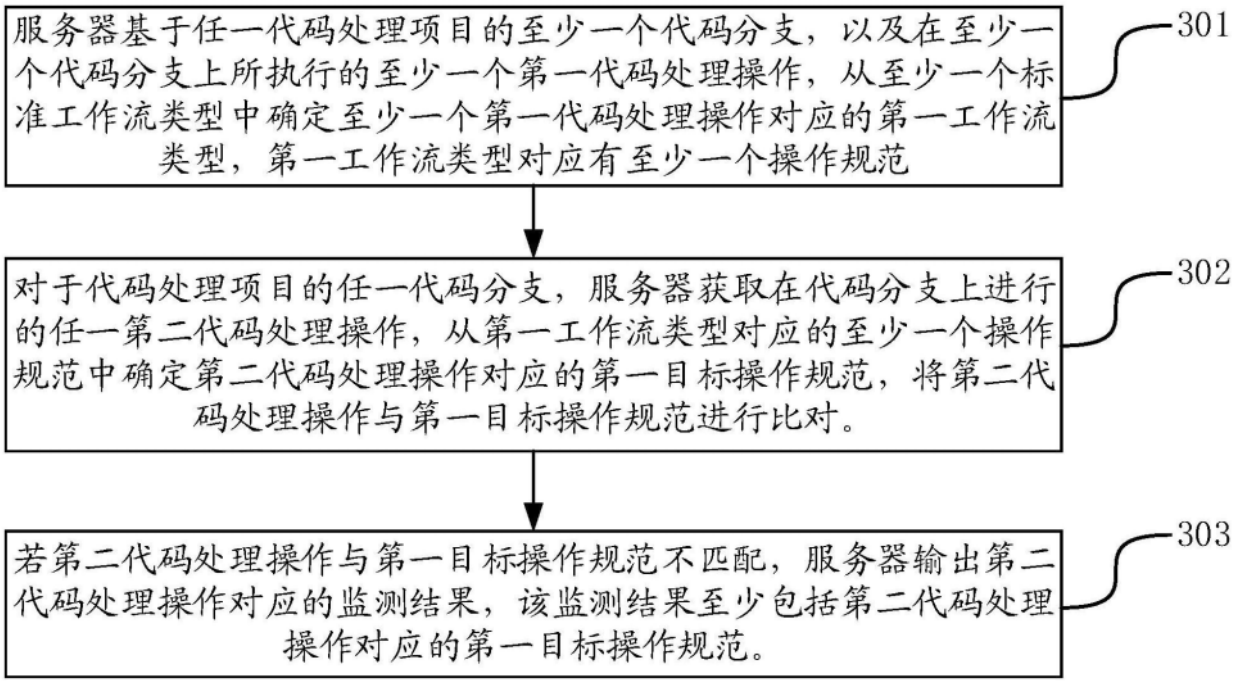

服务器基于任一代码处理项目的至少一个代码分支，以及在至少一个代码分支上所执行的至少一个第一代码处理操作，从至少一个标准工作流类型中确定至少一个第一代码处理操作对应的第一工作流类型，第一工作流类型对应有至少一个操作规范
301
对于代码处理项目的任一代码分支，服务器获取在代码分支上进行的任一第二代码处理操作，从第一工作流类型对应的至少一个操作规范中确定第二代码处理操作对应的第一目标操作规范，将第二代码处理操作与第一目标操作规范进行比对。
302
若第二代码处理操作与第一目标操作规范不匹配，服务器输出第二代码处理操作对应的监测结果，该监测结果至少包括第二代码处理操作对应的第一目标操作规范。
303

图3

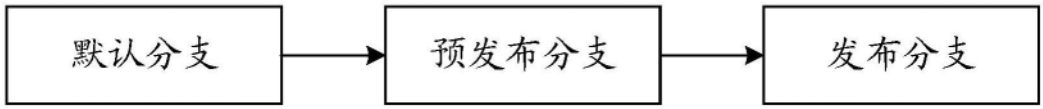

默认分支
预发布分支
发布分支

图4

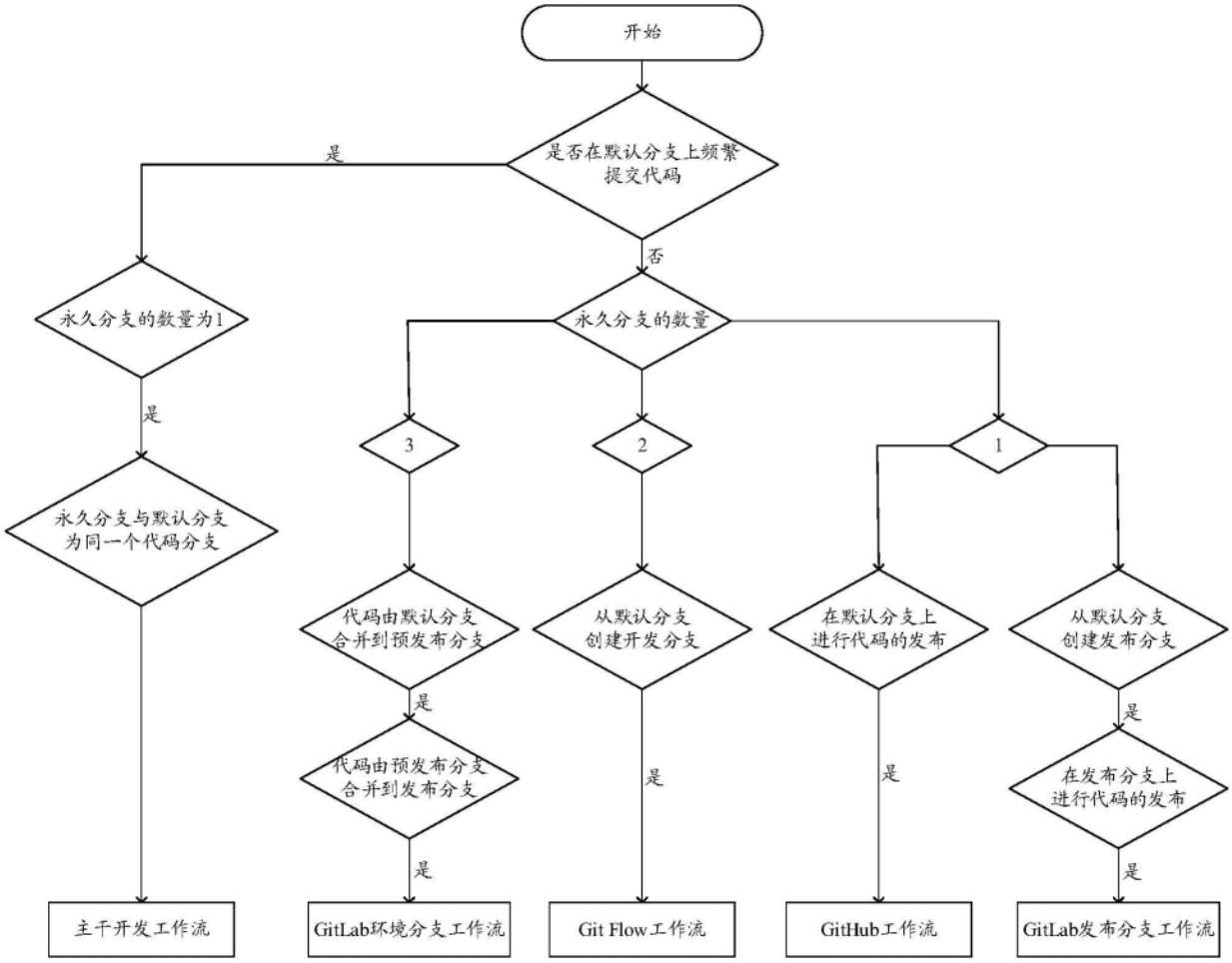

开始
是否在默认分支上频繁提交代码
是
否
永久分支的数量为1
是
永久分支与默认分支为同一个代码分支
主干开发工作流
永久分支的数量
3
2
1
代码由默认分支合并到预发布分支
是
代码由预发布分支合并到发布分支
是
GitLab环境分支工作流
从默认分支创建开发分支
是
Git Flow工作流
在默认分支上进行代码的发布
是
GitHub工作流
从默认分支创建发布分支
是
在发布分支上进行代码的发布
是
GitLab发布分支工作流

图5

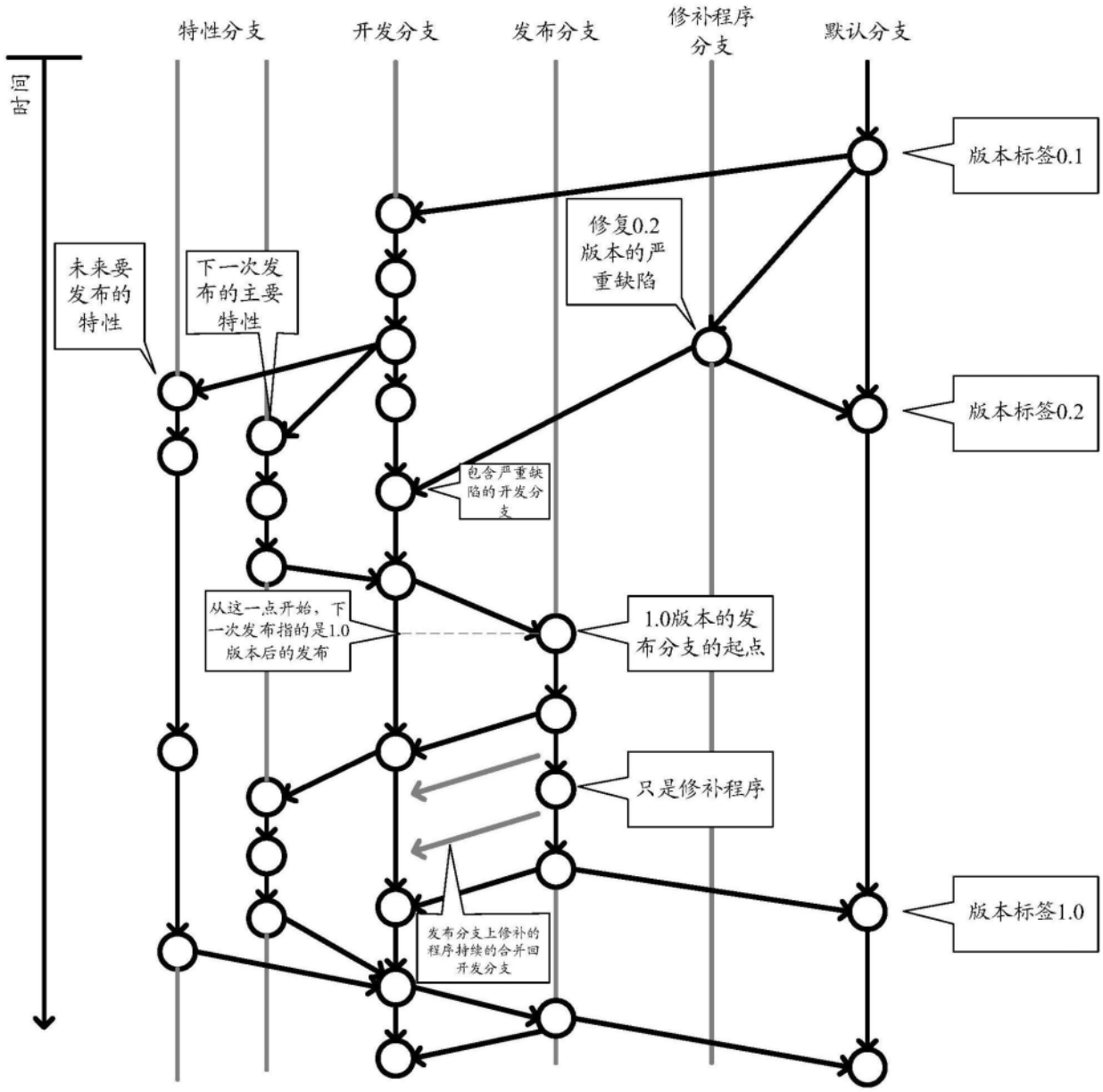

特性分支
开发分支
发布分支
修补程序分支
默认分支
时间
版本标签0.1
修复0.2版本的严重缺陷
未来要发布的特性
下一次发布的主要特性
版本标签0.2
包含严重缺陷的开发分支
从这一点开始，下一次发布指的是1.0版本后的发布
1.0版本的发布分支的起点
只是修补程序
版本标签1.0
发布分支上修补的程序持续的合并回开发分支

图6

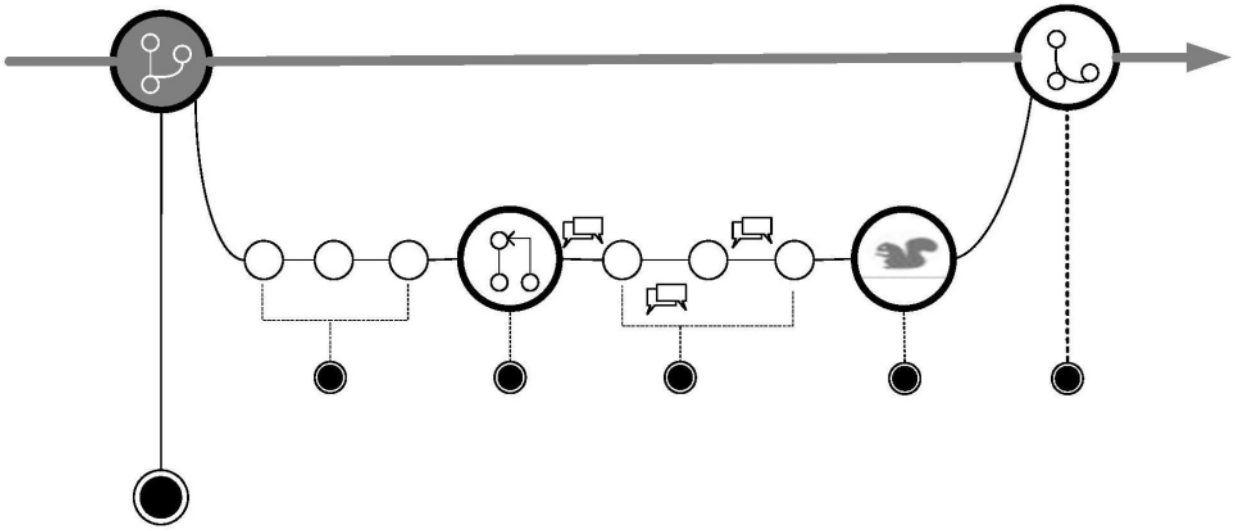


图7

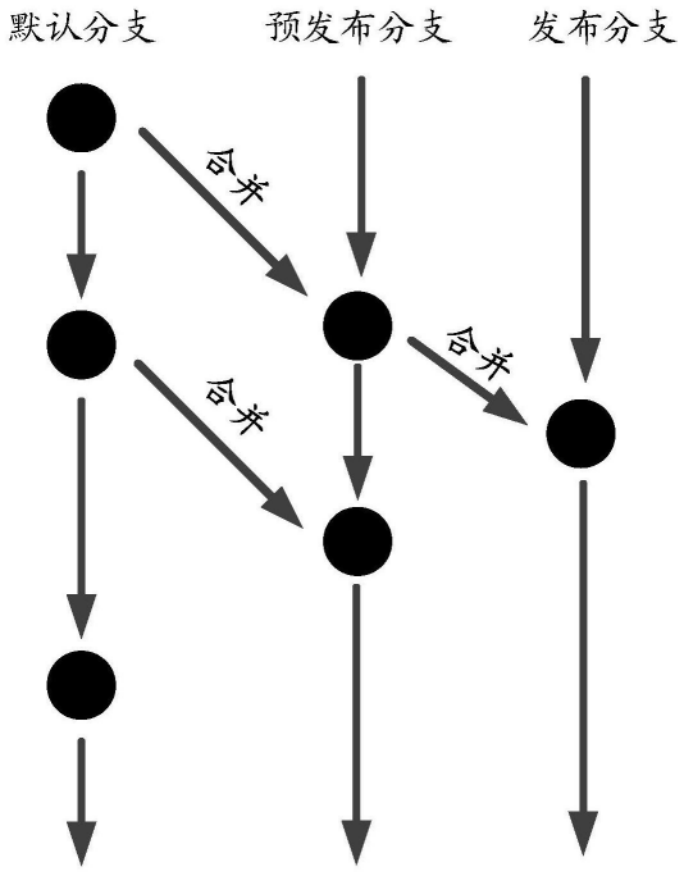


图8

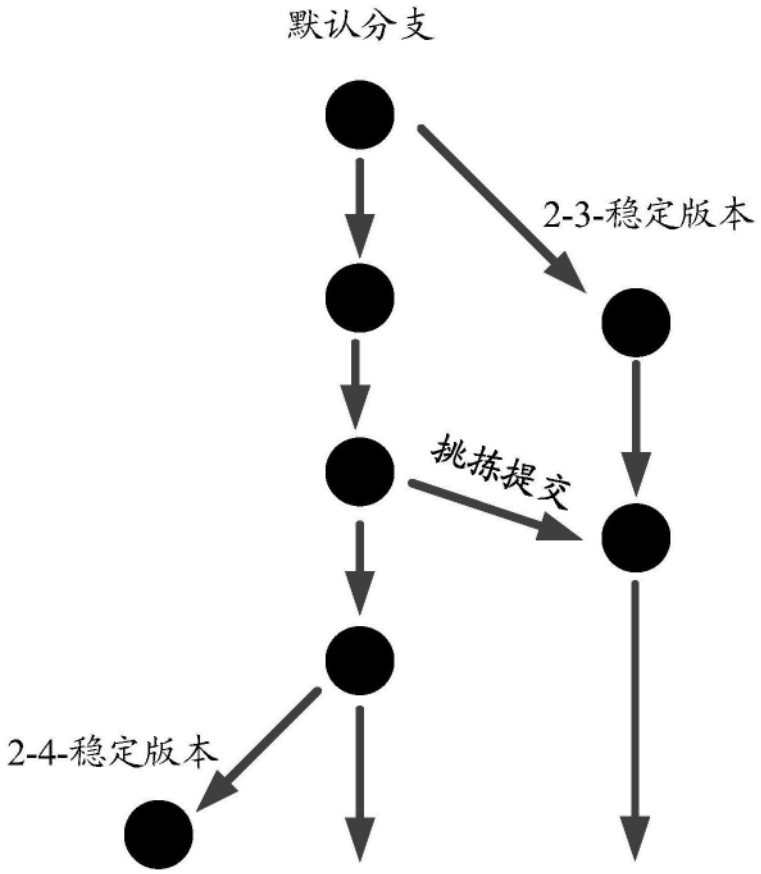


图9

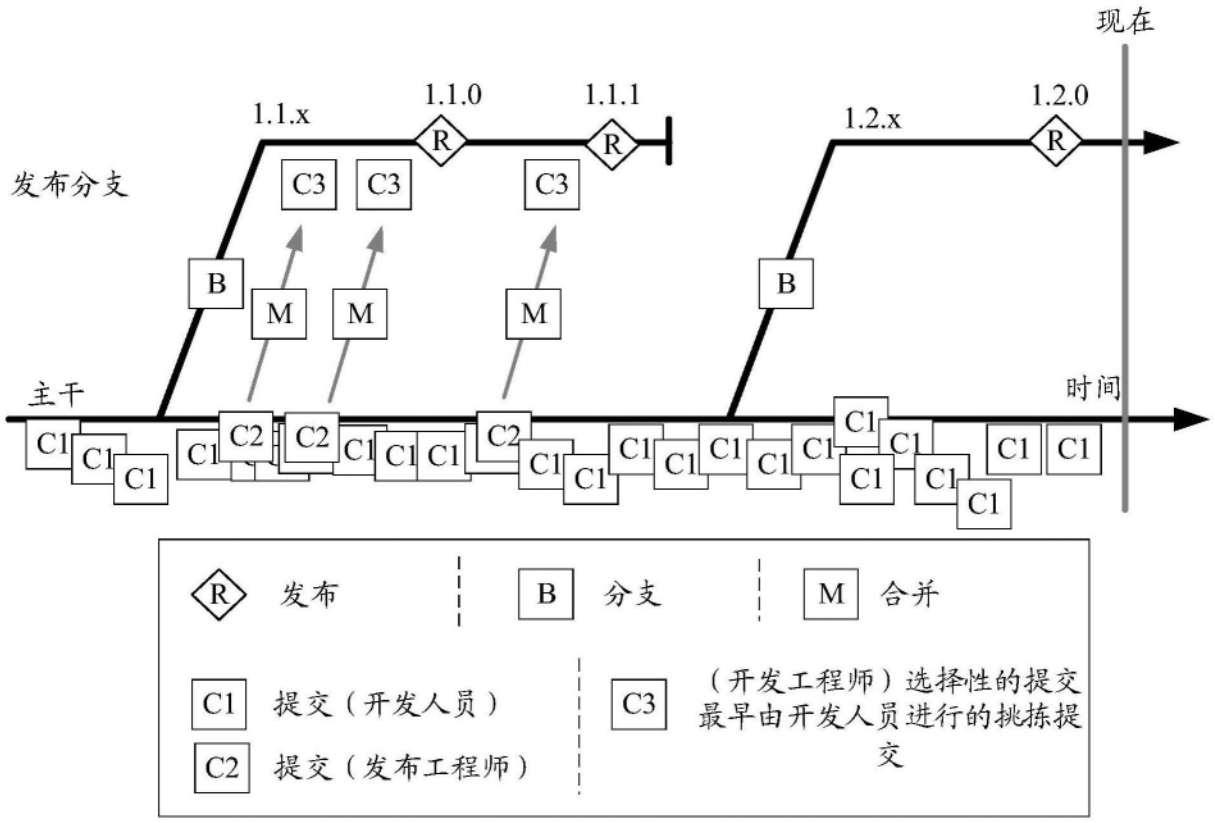

现在
1.1.x
1.1.0
1.1.1
1.2.x
1.2.0
发布分支
主干
时间
B
M
C3
C2
C1
R
发布
分支
合并
提交（开发人员）
提交（发布工程师）
（开发工程师）选择性的提交最早由开发人员进行的挑拣提交

图10

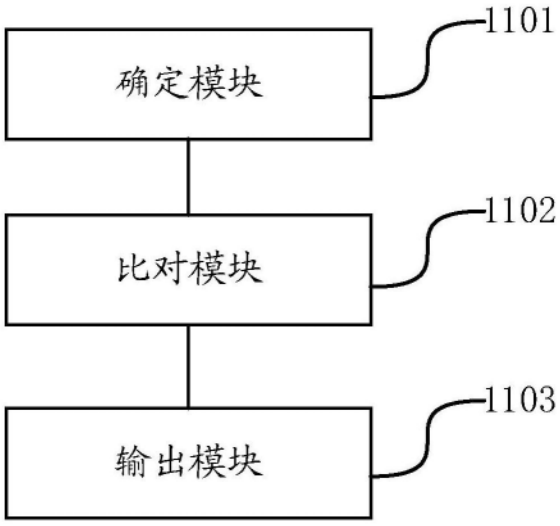

1101
确定模块
1102
比对模块
1103
输出模块

图11

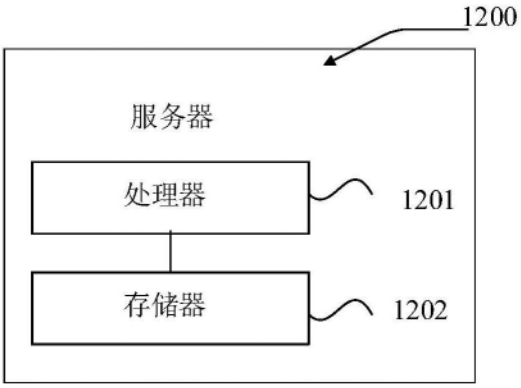


图12