[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 11/36 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200610135534.0

[43] 公开日 2008 年 4 月 23 日

[11] 公开号 CN 101165658A

[22] 申请日 2006.10.18
[21] 申请号 200610135534.0
[71] 申请人 国际商业机器公司
地址 美国纽约
[72] 发明人 尹 俊 黄省江 史蒂文·阿特金

[74] 专利代理机构 北京市中咨律师事务所
代理人 于 静 张亚非

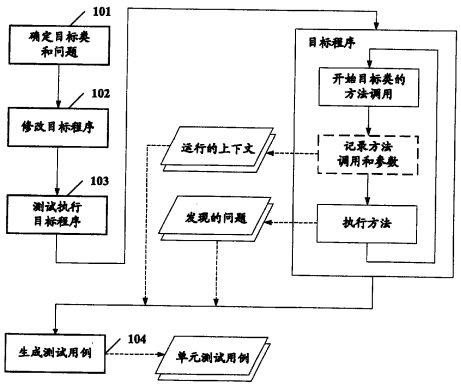
权利要求书 7 页 说明书 26 页 附图 8 页

[54] 发明名称

自动生成可再现运行时问题的单元测试用例的方法和系统

[57] 摘要

本发明公开了一种用于自动生成可再现运行时问题的计算机程序单元测试用例的方法和系统。该方法包括：根据计算机程序中所关注的一个或多个目标程序单元和可能发生的运行时问题修改该程序；测试执行该修改后的程序；以及根据在所关注的目标程序单元的执行中出现的所关注的运行时问题自动生成单元测试用例。其中所述修改步骤在程序中加入捕获代码和问题检测代码，捕获代码被配置为用于记录该程序中所关注的目标程序单元的执行路径及执行上下文，而问题检测代码被配置为用于检测该程序中所关注的目标程序单元的执行所可能发生的运行时问题。本发明还提供了使用上述方法及系统进行调试和回归测试的方法及系统，以及计算机程序测试方法及系统。



1. 一种用于自动生成可再现运行时问题的计算机程序单元测试用例的方法，包括以下步骤：

根据计算机程序中所关注的一个或多个目标程序单元和可能发生的运行时问题修改该程序；

测试执行该修改后的程序；以及

根据在所述所关注的目标程序单元的执行中出现的所述所关注的运行时问题自动生成单元测试用例。

2. 根据权利要求 1 的方法，其中所述运行时问题包括非期望的异常和预定行为规则的违反。

3. 根据权利要求 2 的方法，还包括确定所述程序中所述所关注的目标程序单元以及所述所关注的非期望异常和/或预定行为规则的违反的步骤。

4. 根据权利要求 3 的方法，还包括定义行为规则的步骤。

5. 根据权利要求 4 的方法，其中所述定义行为规则的步骤包括：

确定包含目标方法和检验代码的占位符的规则模板；

确定目标方法；以及

开发检验代码。

6. 根据权利要求 1－5 中任何一个的方法，其中所述修改步骤包括：

在所述程序中加入捕获代码，所述捕获代码被配置为用于记录该程序中所述所关注的目标程序单元的执行路径及执行上下文；以及

在该程序中加入问题检测代码，所述问题检测代码被配置为用于检测该程序中所述所关注的目标程序单元的的执行所可能抛出的所述所关注的非期望异常以及所可能产生的所述所关注的预

定行为规则的违反。

7. 根据权利要求 6 的方法，其中所述计算机程序为面向对象的程序，所述程序单元为类，所述执行路径为所述目标类的各实例对象的每一次外部方法调用的序列，并且所述执行上下文为所述目标类的各实例对象的每一次外部方法调用的参数的值和返回值。

8. 根据权利要求 7 的方法，其中所述捕获代码进一步被配置为用于记录所述目标类的各实例对象的调用参数中不可被复制的对象在目标类中的被调用方法、调用参数及其返回值，并且所述自动生成单元测试用例的步骤包括：

根据所述捕获代码所记录的所述目标类的各实例对象的调用参数中不可被复制的对象在目标类中的被调用方法、调用参数及其返回值生成所述不可被复制的对象的模拟对象；以及

根据所述捕获代码所记录的相关的执行路径及执行上下文以及所述生成的模拟对象生成单元测试用例。

9. 根据权利要求 7 的方法，其中，所述问题检测代码还被配置为用于每当检测到所述所关注的非期望异常或预定行为规则的违反时，就触发所述自动生成单元测试用例的步骤根据所述捕获代码所记录的相关的执行路径及执行上下文生成单元测试用例。

10. 根据权利要求 7 的方法，其中，所述问题检测代码还被配置为用于每当检测到所述非期望异常或预定行为规则的违反时，将所述所关注的非期望异常或预定行为规则的违反记入问题列表，并且所述自动生成单元测试用例的步骤包括当所述测试执行过程结束后，根据所述问题列表以及所述捕获代码所记录的相关的执行路径及执行上下文生成单元测试用例。

11. 根据权利要求 9 或 10 的方法，其中所述生成的单元测试用例包括所述目标类的实例对象从被构造到被检测到所述非期望异常或预定行为规则的违反为止的外部方法调用的序列，以及每

一次外部方法调用的参数的值。

12. 根据权利要求 10 的方法，其中所述捕获代码进一步被配置为用于记录所述目标类的各实例对象的调用参数中不可被复制的对象在目标类中的被调用方法、调用参数及其返回值，并且其中所述根据所述问题列表以及所述捕获代码所记录的相关的执行路径及执行上下文生成单元测试用例的步骤包括以下子步骤：

遍历所述问题列表中的每一项非期望异常或预定行为规则的违反；

针对所遍历的每一项非期望异常或预定行为规则的违反，访问所述捕获代码所记录的相应的执行路径和执行上下文以及所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值；

根据所访问的所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值生成所述不可被复制的对象的模拟对象；以及

根据所访问的相应的执行路径和执行上下文以及所生成的模拟对象构造单元测试用例。

13. 一种用于自动生成可再现运行时问题的计算机程序单元测试用例的方法，包括以下步骤：

根据该计算机程序的测试后生成的测试缺陷描述，确定缺陷的性质及发生缺陷的程序单元；

根据所确定的缺陷的性质及发生缺陷的程序单元修改该程序；

测试执行该程序；以及

根据检测到的所述缺陷以及所述捕获代码记录的相关的执行路径及上下文生成单元测试用例。

14. 根据权利要求 13 的方法，其中所述修改步骤包括：

在该程序中加入捕获代码，所述捕获代码被配置为用于记录

该程序中的所述程序单元的执行路径及执行上下文；以及

在该程序中加入问题检测代码，所述问题检测代码根据所确定的缺陷的性质被配置为用于检测该程序中所述程序单元的执行中的所述缺陷。

15. 一种计算机程序测试方法，包括以下步骤：

在所述程序中加入捕获代码和问题检测代码，其中所述捕获代码被配置为用于记录该程序中所关注的目标程序单元的执行路径及执行上下文，而所述问题检测代码被配置为用于检测并记录该程序中所述所关注的程序单元的执行所可能抛出的所关注的非期望异常以及所可能产生的所关注的预定行为规则的违反；以及

测试执行经过所述加入的该程序，以检测并记录所述所关注的非期望异常以及预定行为规则的违反。

16. 根据权利要求 15 的方法，还包括定义所述预定行为规则的步骤。

17. 一种用于自动生成可再现运行时问题的计算机程序单元测试用例的系统，包括：

修改模块，用于根据计算机程序中所关注的一个或多个目标程序单元和可能发生的运行时问题修改该程序；

测试执行模块，用于测试执行所述被修改后的程序；以及

单元测试用例生成模块，用于根据在所述所关注的程序单元的执行中出现的所述所关注的运行时问题自动生成单元测试用例。

18. 根据权利要求 17 的系统，其中所述运行时问题包括非期望的异常和预定行为规则的违反。

19. 根据权利要求 18 的系统，还包括确定模块，用于确定所述程序中所述所关注的目标程序单元以及所述所关注的非期望异常和/或预定行为规则的违反。

20. 根据权利要求 19 的系统，还包括规则定义模块，用于定

义可由所述确定模块确定的规则。

21. 根据权利要求 20 的系统，其中所述规则定义模块被配置为用于通过以下步骤来定义规则：

确定包含目标方法和检验代码的占位符的规则模板；

确定目标方法；以及

开发检验代码。

22. 根据权利要求 17－20 中任何一个的系统，其中所述修改模块被配置为用于：

在该程序中加入捕获代码，所述捕获代码被配置为用于记录该程序中所述所关注的程序单元的执行路径和执行上下文；以及

在该程序中加入问题检测代码，所述问题检测代码被配置为用于检测所述程序单元的执行所可能抛出的所关注的非期望异常以及所可能产生的所关注的预定行为规则的违反。

23. 根据权利要求 22 的系统，其中所述计算机程序为面向对象的程序，所述程序单元为类，所述执行路径为所述目标类的各实例对象的每一次外部方法调用的序列，并且所述执行上下文为所述目标类的各实例对象的每一次外部方法调用的参数的值和返回值。

24. 根据权利要求 23 的系统，其中所述捕获代码进一步被配置为用于记录所述目标类的各实例对象的调用参数中不可被复制的对象在目标类中的被调用方法、调用参数及其返回值，并且所述单元测试用例自动生成模块被配置为用于：

根据所述捕获代码所记录的所述目标类的各实例对象的调用参数中不可被复制的对象在目标类中的被调用方法、调用参数及其返回值生成所述不可被复制的对象的模拟对象；以及

根据所述捕获代码所记录的相关的执行路径及执行上下文以及所述生成的模拟对象生成单元测试用例。

25. 根据权利要求 23 的系统，其中，所述问题检测代码还被

配置为用于每当检测到所述非期望异常或预定行为规则的违反时，就触发所述单元测试用例生成模块根据所述捕获代码所记录的相关的执行路径及执行上下文生成单元测试用例。

26. 根据权利要求 23 的系统，其中，所述问题检测代码还被配置为用于每当检测到所述非期望异常或预定行为规则的违反时，将所述非期望异常或预定行为规则的违反记入问题列表，并且所述单元测试用例生成模块被配置为用于当所述测试执行模块进行的测试执行过程结束后，根据所述问题列表以及所述捕获代码所记录的相关的执行路径及执行上下文生成单元测试用例。

27. 根据权利要求 25 或 26 的系统，其中所述生成的单元测试用例包括目标类的实例对象从被构造到被检测到所述非期望异常或预定行为规则的违反为止的外部方法调用的序列，以及每一次外部方法调用的参数的值。

28. 根据权利要求 26 的系统，其中所述捕获代码进一步被配置为用于记录所述目标类的各实例对象的调用参数中不可被复制的对象在目标类中的被调用方法、调用参数及其返回值，并且其中所述单元测试用例生成模块还包括：

问题列表访问模块，用于遍历所述问题列表中的每一项非期望异常或预定行为规则的违反；

执行路径及上下文访问模块，用于针对所遍历的每一项非期望异常或预定行为规则的违反，访问所述捕获代码所记录的相应的执行路径和执行上下文以及所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值；

模拟对象构造模块，用于根据所访问的所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值生成所述不可被复制的对象的模拟对象；以及

单元测试用例构造模块，用于根据所访问的相应的执行路径和执行上下文以及所生成的模拟对象构造单元测试用例。

29. 一种计算机程序测试装置，包括：

修改模块，用于在所述程序中加入捕获代码和问题检测代码，所述捕获代码被配置为用于记录该程序中所关注的目标程序单元的执行路径及执行上下文，而所述问题检测代码被配置为用于检测并记录该程序中所述所关注的程序单元的执行所可能抛出的所关注的非期望异常以及所可能产生的所关注的预定行为规则的违反；以及

测试执行模块，用于测试执行经过所述加入的该程序，以检测并记录所关注的目标程序单元的执行中出现的所述所关注的非期望异常以及预定行为规则的违反。

30. 根据权利要求 29 的装置，还包括用于定义所述预定行为规则的规则定义模板。

31. 一种包含计算机可读介质的计算机程序产品，该计算机可读介质具有包含在其中的用于使得具有数据处理能力的系统执行根据前述方法权利要求中任何一个的方法的步骤的程序指令。

自动生成可再现运行时问题的单元测试用例的方法和系统

技术领域

本发明涉及计算机程序的测试及调试领域，更具体地，涉及一种用于自动生成可再现运行时问题的计算机程序单元测试用例的方法和系统。

背景技术

软件产品例如 IBM 的软件产品的开发通常遵循这样的过程，开发团队负责产品开发和单元测试，产品其他的验证测试工作则由专门的测试团队来负责，验证测试所发现的缺陷由开发团队来更正。单元测试是白盒测试，主要目的是验证代码逻辑，而验证测试则是黑盒测试，用来验证软件功能和表现。

在验证测试期间，可能会发现很多运行时问题，例如非期望的异常以及不正确的或不正常的程序行为。由于测试团队和开发团队是相互独立的，验证测试环境和开发环境也不相同，验证测试中发现的缺陷往往只能通过测试者的文字描述和软件本身记录的日志信息反馈给开发团队。由于测试环境缺乏调试工具，而缺陷描述和日志信息又无法直接调试，这样开发团队必须先根据缺陷描述重建测试环境来进行调试。这通常是非常繁琐和低效的过程。并且有时非常难以再现运行时问题并发现它们的原因。

而且，当开发团队修正了当前发现的缺陷后或者开发了软件新的版本后，因为加入了新的代码，测试团队仍然必须确保之前发现的并已修正的所有缺陷都仍然是被修正的，没有受到当前修改的影响，即进行所谓的回归测试。虽然测试团队必须进行回归测试，但是如果开发团队已经能够提前确保回归测试的正确性，那么测试团队则只要再做一次验证而已，无疑回归测试的效率将会大大提高。因为开发团队验证工具通常是单元测试用

例，所以就要求开发团队为每一个已发现的缺陷开发出能够验证其是否被修正的单元测试用例。但是由于前一个问题的原因，调试工作已然复杂，根据缺陷开发单元测试用例往往也是很繁琐的过程，常常被开发团队所忽略。

目前的测试自动化工具，例如 Rational Robot 和 WinRunner 等，仅仅针对测试团队和测试过程，其出发点是替代测试者重复的测试动作。它们致力于通过记录和回放用户动作，例如键击和鼠标点击，而不是捕获内部程序行为。它们生成脚本以模拟来自用户和运行时环境的输入。难以确定必须记录多少用户交互以便再现问题。并且由于回放脚本中可能包含一些不相关的动作，再现问题所需要的时间可能不是最佳的。

单元测试目前已在软件开发过程中得到广泛使用。开发者可以在将程序提交测试团队进行验证测试之前通过执行单元测试套件（test suite）来迅速和简单地发现和消除缺陷。一般来说，单元测试的成本远低于验证测试的成本。

目前也存在几种技术和单元测试工具，例如 JTest，能够帮助开发者生成单元测试用例。然而，它们只是针对开发团队和开发过程，是根据对代码的静态分析等手段而不是运行时上下文来生成测试用例的。但是单纯的静态分析不能确定程序执行时各程序单元执行的上下文，不能利用运行时上下文和状态创建出可再现运行时问题的测试用例。因此，它们都不能连接测试过程和开发过程，来解决上述问题。

因此，如果能够根据运行时刻的缺陷自动生成单元测试用例，以上两个问题将得以有效的解决：一方面开发团队可以直接使用该测试用例进行调试，省去重建测试环境的工作；另一方面开发团队还可以将此测试用例加入单元测试过程，以确保回归测试的正确性。

标题为“用于监视程序执行的方法和装置”（Method And Apparatus For Monitoring The Execution Of A Program）的美国专利 6,738,778 公开了一种跟踪调试工具，它可以在程序中被调用，能够自动收集并打印程序运行信息（如程序当前运行在哪个类的哪一行）。然而，此专利只是打印

用于调试的跟踪信息，而不是生成单元测试用例。也不能主动发现程序运行中的错误和异常，并以此触发生成单元测试用例的。

标题为“用于在软件开发中集成自动软件测试”（Method For Integrating Automated Software Testing With Software Development）的美国专利 6,067,639 公开了一种需要集成在开发过程中的自动化软件测试方法。该方法需要软件开发者在开发阶段就同时开发一些测试操作对象（Test Operation Object），一个测试操作对象中实现对程序的某种操作的调用。这样在测试时，软件测试者就可以编写自动测试程序或脚本，通过调用这些被测试程序中的测试操作对象来自动化测试软件。由于这些对象提供了统一的接口（Interface），这样测试者就不需要为不同的测试环境开发不同的自动测试程序或脚本。然而，该专利需要软件开发者根据统一的接口开发测试操作对象，实际上相当于在开发测试用例，而不是自动生成单元测试用例。并且，它要求开发者在开发程序时就一起开发测试操作对象。

从以上所述可见，本领域中显然需要一种能够根据运行时刻的缺陷自动生成单元测试用例的方法和系统。

发明内容

为解决现有技术中由于无法在测试环境中生成单元测试用例而导致的难以在调试时再现运行时问题及便利于回归测试的上述问题并带来其他益处，而做出了本发明。

根据本发明的一个方面，提供了一种用于自动生成可再现运行时问题的计算机程序单元测试用例的方法，该方法包括以下步骤：根据计算机程序中所关注的一个或多个目标程序单元和可能发生的运行时问题修改该程序；测试执行该修改后的程序；以及根据在所述所关注的目标程序单元的执行中出现的所述所关注的运行时问题自动生成单元测试用例。

根据本发明的另一个方面，提供了一种用于自动生成可再现运行时问题的计算机程序单元测试用例的方法，该包括以下步骤：根据该计算机程

序的测试后生成的测试缺陷描述，确定缺陷的性质及发生缺陷的程序单元；根据所确定的缺陷的性质及发生缺陷的程序单元修改该程序；测试执行该程序；以及根据检测到的所述缺陷以及所述捕获代码记录的执行路径及上下文生成单元测试用例。

根据本发明的另外一个方面，提供了一种计算机程序测试方法，该方法包括以下步骤：在所述程序中加入捕获代码和问题检测代码，其中所述捕获代码被配置为用于记录该程序中所关注的目标程序单元的执行路径及执行上下文，而所述问题检测代码被配置为用于检测并记录该程序中所述所关注的程序单元的执行所可能抛出的所关注的非期望异常以及所可能产生的所关注的预定行为规则的违反；以及测试执行经过所述加入的该程序，以检测并记录所述所关注的非期望异常以及预定行为规则的违反。

根据本发明的进一步的方面，还提供了一种用于自动生成可再现运行时问题的计算机程序单元测试用例的系统，该系统包括：修改模块，用于根据计算机程序中所关注的一个或多个目标程序单元和可能发生的运行时问题修改该程序；测试执行模块，用于测试执行所述被修改后的程序；以及单元测试用例生成模块，用于根据在所述所关注的程序单元的执行中出现的所述所关注的运行时问题自动生成单元测试用例。

根据本发明的再一个方面，提供了一种计算机程序测试装置，该装置包括：修改模块，用于在所述程序中加入捕获代码和问题检测代码，所述捕获代码被配置为用于记录该程序中所关注的目标程序单元的执行路径及执行上下文，而所述问题检测代码被配置为用于检测并记录该程序中所述所关注的程序单元的执行所可能抛出的所关注的非期望异常以及所可能产生的所关注的预定行为规则的违反；以及测试执行模块，用于测试执行经过所述加入的该程序，以检测并记录所关注的目标程序单元的执行中出现的所述所关注的非期望异常以及预定行为规则的违反。

根据本发明的另外一个方面，提供了一种包含计算机可读介质的计算机程序产品，该计算机可读介质具有包含在其中的用于使得具有数据处理能力的系统执行根据前述方法权利要求中任何一个的方法的步骤的程序指

令。

从以上所述可见，本发明的主要目是帮助开发团队修正缺陷以及确保回归测试的正确性，以此来提高软件开发和测试的效率。同时，由于本发明可以定义产生测试用例的行为规则，因此也有可能在测试过程中发现尚未表现出的潜在的缺陷，从而帮助开发团队或者测试团队发现缺陷。

本发明所带来的主要益处包括:

可根据运行时上下文生成单元测试用例以再现运行时问题，这样，由于排除了不相关的操作，再现运行时问题所需的执行时间大为减少，从而便利了对程序的调试;

开发者仅需要围绕所述单元测试用例工作，而无需创建验证测试中的复杂环境，从而大大节约了时间和努力;

当施加了新的代码更新后，开发者可使用单元测试用例来简化对旧问题的验证；以及

有助于发现潜在的缺陷，即有助于对程序的验证测试。

附图说明

所附权利要求中阐述了被认为是本发明的特点的新颖特征。但是，通过在结合附图阅读时参照下面对说明性实施例的详细说明将更好地理解发明本身以及其优选使用模式、另外的目标以及优点，其中:

图 1 示出了根据本发明的实施例用于为计算机程序自动生成可再现运行时问题的单元测试用例的方法的示意性流程图;

图 2 示出了根据本发明的实施例的用于检测非期望的异常的过程的示意性流程图;

图 3 示出了根据本发明的实施例的用于检测预定行为规则的违反的过程的示意性流程图;

图 4 示出了在被测试程序的执行过程中一对象的交互过程的示意图;

图 5 示出了根据本发明的一个实施例的图 1 所示方法中根据问题列表以及执行路径和执行上下文生成单元测试用例步骤中的具体步骤的流程

图；

图 6 示出了根据本发明的另一个实施例的用于自动生成可再现运行时问题的计算机程序单元测试用例的方法的流程图；

图 7 示出了根据本发明的实施例的用于自动生成可再现运行时问题的计算机程序单元测试用例的系统的框图；以及

图 8 示出了根据本发明的进一步的实施例的单元测试用例生成模块的结构示意图。

具体实施方式

本发明公开了一种可自动生成可再现程序运行时问题的单元测试用例的方法和系统。该系统或方法使用加入的检测代码来检测预先确定的运行时问题，例如非期望的异常和非正常的程序行为。同时，该系统或方法使用加入的捕获代码来收集所确定对象的执行上下文和其执行路径。当在测试过程中发生问题时，该系统或方法可生成该对象的单元测试用例。该单元测试用例可通过以所记录的上下文重放该对象的执行路径来再现该问题。

下面参照附图描述本发明的实施例。然而，应当理解的是，本发明并不限于所介绍的特定实施例。在附图及以下描述中所给出的大量细节只是为了说明之用，以便使本领域的技术人员能充分理解本发明的基本思想和实现本发明，而不构成对本发明的限制。

图 1 示出了根据本发明的实施例用于为计算机程序例如 Java 程序自动生成可再现运行时问题的单元测试用例的方法的一般步骤。

在步骤 101 中，确定所关注的目标程序单元和可能发生的运行时问题。对于面向对象的程序例如 Java 程序来说，所述目标程序单元为目标类。步骤 101 优选地在开发环境中执行。该步骤既可以在 GUI 工具中进行，即用户通过 GUI 工具确定目标类和运行时问题，而由 GUI 工具根据用户的确定生成将提供给下一步骤的包含确定的目标类和运行时问题的配置信息；也可以由用户手工进行，即由用户手工编制将提供给下一步骤的包含确定

的目标类和运行时问题的配置信息；也可以与将在下面描述的用于修改目标程序的下一个步骤组合在一起，即在修改目标程序时确定所关注的目标类和运行时问题，也就是所关注的目标类和运行时问题体现在如何以及在何处对目标程序进行修改中，并且在这种情况下，该方法中可以没有该确定步骤 101。

目标类是需要进行单元测试的类。目标类的所有对象将被跟踪以生成单元测试用例。目标类可通过其限定名被确定。

所述运行时问题可以是非期望的异常或预定行为规则的违反。非期望的异常是由对象方法抛出的未经检查的运行时异常，例如 Java 程序中的 java.lang.NullPointerException，即程序某处试图在一个 null 对象上调用对象方法。它们往往是缺陷产生的原因。非期望的异常可以通过方法的限定名和异常的类别来确定。预定的行为规则描述了所确定对象的正常行为，包括正确的方法调用顺序、有效的方法参数和返回值等。

在一优选实施例中，在步骤 101 之前，还包括定义行为规则的步骤，以便定义将在步骤 101 中确定的预定行为规则。该定义行为规则的步骤优选地在开发环境中执行。一种定义规则的方法是通过对每一条规则确定规则模板、目标方法以及开发检验代码。规则模板是指预先定义好的一些模式，例如“Never call <X> when <CONDITION>”, “Never call <X> before <Y> of an object when <CONDITION>”等等。目标方法即是定义模板中的“<X>”、“<Y>”具体是什么类的什么方法，如确定 <X> 是 java.lang.String.charAt(int)，这决定了检验代码插入的位置。检验代码即是在目标方法被调用的前后对方法的对象、参数、返回值等等实现模板中的“<CONDITION>”检验。在本发明的一个优选实施例中，检验代码通过继承一个规则抽象类来实现，如对于上述模板“Never call <X> when <CONDITION>”需要继承 AbstractNeverCallXRule 类，并重载其中的 matchedX 方法即可针对取得的上下文进行检验。规则的抽象类是模板的一部分，通过它隐藏了模板的具体实现，从而开发规则时可以只专注于检验代码的开发。

应指出，通过规则模板以及规则的抽象类来定义待确定的规则的方法只是本发明的定义和确定规则的一种实现方式，本发明也可采用其他定义和确定规则的方式，例如通过直接插入规则的检验代码（定义和确定规则的步骤隐含在插入检验代码的过程中）等等。

作为规则的示例，例如，Java 全球化功能中可以有码点处理（Code Point Handling）、字符串处理（String Handling）、编码（Encoding）等规则。作为一个更具体的示例，java.lang.String.substring(int beginIndex, int endIndex)方法用于从一个字符串中取出一段子字符串，它简单地从一个 16 位的 char 类型数组 char[]中取出由 beginIndex 和 endIndex 限定的子 char[]。然而由于有一些 Unicode 补充字符是由 2 个 16 位的 char 表示的（Surrogate），比如字符"\uD840\uDC00"表示一补充汉字，此外某些文字中的组合字符，比如字符"À"，也可以使用比如"\u0041\u0300"表示（字符 A 和组合字符一瞥），所以在使用 String.substring 时，很有可能取出的子串含有不完整的字符表示。于是可以定义这样一条规则，在程序中使用到的所有 String.substring 之前检验传入的参数和字符串本身，如果发现 beginIndex 和 endIndex 正位于一个不完整的字符上，则会触发一个规则违例。这条规则还可以定义得更宽，规定只要字符串含有多 char 表示的字符，就不能直接使用 substring 方法，而不论 beginIndex 和 endIndex 是否会取得不完整字符，因为它可导致潜在的缺陷。

在步骤 102，根据确定的目标类和运行时问题修改目标程序，即在目标程序中加入捕获代码（图 1 中的虚线框部分）和问题检测代码（图 2 和图 3 中的虚线框部分）。优选地，步骤 102 在开发环境中执行。

在本发明的一优选实施例中，在编译后的 Java 字节码程序中，加入所述捕获代码和问题检测代码。可使用字节码插入工具来修改目标程序的类文件，以加入捕获代码和问题检测代码。但是在本发明的其他实施例中，如下文中所述，提供了可以在源代码中被调用的方法，来人工或通过工具加入所述捕获代码和问题检测代码以触发测试用例的生成。

捕获代码主要用于记录目标类对象的每一次外部方法调用以及所有方

法调用参数的值，作为目标类对象的执行路径和执行上下文。

有很多修改字节码的方法可以捕捉方法调用的参数和结果，不管是插入在方法调用的地方还是方法体内部。在 JVM 中，方法调用只有一个字节码，传入方法的参数就是在执行方法调用的字节码之前的 JVM 参数栈中的所有对象，在该方法调用字节码执行完成之后，JVM 参数栈会被清空并填上返回值。因此一种简单的获得上下文的方法是替换此方法调用的字节码，使其调用另一个对象的静态方法，此静态方法的参数和返回值类型和原有方法基本相同，只是参数里多出一个 this object。第二种方法是在此调用字节码之前和之后插入字节码以获取栈中对象（参数）并恢复栈。当然还有一种方法就是直接修改目标方法本身的字节码，只需要修改一处。使用直接修改目标方法时，对于某些没有办法静态修改的类，如 JRE 中的类（String 等），必须采用动态修改的方式，即在 JVM 载入后进行修改。

此外，在一个完整的单元测试用例中，除被测试的目标类对象之外，还存在着需要模拟的对象（mocked object）和其他无需模拟的对象，这些是目标类对象的方法调用中的对象参数。对于 Java 程序来说，无需模拟的对象通常是某些 Java API 中的类，它们可被复制。可以确定需要被模拟的对象，如果没有确定被模拟的对象类，那么所有不被识别为无需模拟的对象将会被作为需要被模拟的对象对待。捕获代码还将被加入在对需要模拟的对象的方法的调用的前后，以记录需要被模拟的对象在被测试对象的方法中表现出的行为，即被调用的方法以及其返回值等数据。通过这两部分的记录，可以使用已有的针对 Java 的模拟技术（例如 JMock）来生成模拟对象，进而生成单元测试用例。一段实现捕获代码的伪码可以简单表示如下：

```
// 用于记录被测试对象的执行路径和上下文
class Captor_UnitTestObject {
  // 该方法将被插入在目标对象的方法调用之前
  public static void beforeCall(String className, String methodName,
```

```
String methodSig, Object thisObject, Object[] args) {
        // 记录执行路径
        RuntimeContextRecorder rc = RuntimeContextLogger.
getRuntimeContextRecorder(className, thisObject);
        rc.recordContext(methodName, methodSig, thisObject, args);
        // 注册需要在此方法执行中关注的被模拟的对象
        rc.beginMockingMonitor(args);
    }

    // 该方法将被插入在目标对象的方法调用之后
    public static void afterCall(String className, Object thisObject) {
        // 取消对被模拟对象的关注
        RuntimeContextRecorder rc = RuntimeContextLogger.
getCurrentRuntimeContextRecorder();
        rc.endMockingMonitor();
    }
}

// 用于记录被模拟对象的行为
public class Captor_MockedObject {
    // 该方法将被插入在目标对象的方法调用之前
    public static void beforeCall(String className, String methodName,
String methodSig, Object thisObject, Object[] args) {
        // 记录调用的参数
        ObjectMocker om = RuntimeContextLogger.
getCurrentRuntimeContextRecorder().getMocker();
        om.addRecordBeforeCall(className, methodName,
methodSig, thisObject, args);
```

```
    }

    // 该方法将被插入在目标对象的方法调用之后
    public static void afterCall(String className, Object thisObject, Object
returnedObject, Throwable thrownObject) {
        // 记录调用的结果
        ObjectMocker om = RuntimeContextLogger.
getCurrentRuntimeContextRecorder().getMocker();
            om.addRecordAfterCall(className, thisObject, returnedObject,
thrownObject);
    }
}
```

其中，RuntimeContextLogger 是一个负责管理执行上下文信息的类，它维护了所有关注的对象的上下文信息。RuntimeContextRecorder 是一个负责记录某个关注对象上下文信息的类，通过调用 RuntimeContextLogger. getRuntimeContextRecorder(Class className, Object thisObject)来获得，其 recordContext 方法即可记录当前的上下文信息。BeginMockingMonitor (Object[] args)方法则启动对所有 args 对象的上下文捕获用来生成 mock 对象，endMockingMonitor() 方法结束 mock 对象的上下文捕获。ObjectMocker 负责记录捕获的 mock 对象的上下文。

应指出，上述伪码和各类只是实现本发明的捕获代码的示例，而不构成对本发明的限制。本发明的捕获代码也可以多种其他代码和类来实现。

问题检测代码被加入在每个需要检测的方法的前后。如图 2 所示。某些代码被加入在方法调用之后以检测所关注的例如在步骤 101 中确定的非期望的抛出的异常。如图 3 所示，其他代码被加入在方法调用之前和之后，以在执行上下文例如调用顺序、参数和返回值中发现对所关注的例如在步骤 101 中确定的行为规则的违反，进而由此来触发单元测试用例的生成。

优选地，问题检测代码用于当发现问题时，将其记入一问题列表，以便在测试执行结束后生成测试用例。作为另一种选择，问题检测代码也可用于每当发现一个问题时，即根据该问题以及捕获代码所记录的执行路径和执行上下文生成一测试用例。一段实现上述 substring 规则的检测代码的伪码可以简单表示如下：

```
public class Detective_StringSubstring {
  // 该方法将被插入在 String.subString 的方法调用之前
  public static void beforeCallX(String className, String methodName,
String methodSig, Object thisObject, Object[] args) {
      AbstractNeverCallXRule rule = DetectiveManager.
loadRule("Globalization.CodePointHandling.StringSubString",thisObject);
      // rule.matchedX 中含有检验代码
      if (rule.matchedX(thisObject, args)) {
          DetectiveManager.addViolation(rule);
          DetectiveManager.generateTestCase();
      }
  }
}
```

其中， AbstractNeverCallXRule 是一种规则的抽象类，用于隐藏规则的具体实现。DetectiveManager 用来载入各种规则类，通过对规则类中方法的调用来检验上下文，然后决定是否出发 generateTestCase()。

应指出，上述伪码和各类只是实现本发明的检测代码的示例，而不构成对本发明的限制。本发明的检测代码也可以多种其他代码和类来实现。

优选地，将在步骤 102 修改的目标程序提供给测试团队，以在测试环境中进行测试。

在步骤 103 中，测试执行目标程序。优选地，步骤 103 在测试环境中执行。

在测试执行过程中，在目标类的每个方法调用之前，捕获代码捕获该调用和该调用的参数。然后执行原来的方法调用。在方法调用的执行过程中，检测代码试图发现非期望和异常和行为规则的违反，并优选地将其记入问题列表。

在步骤 104 中，生成单元测试用例。

在目标程序终止后，系统可使用所记录的执行路径和执行上下文以及问题列表生成单元测试用例。

作为另一种选择，也可在目标程序执行过程中，每当检测到一个所关注的问题，就根据该问题以及所记录的执行路径和执行上下文生成一个测试用例。

对于 Java 程序，可将单元测试用例形成为包含用于 JUnit 的测试用例类的 Java 源文件。

所生成的单元测试用例描述了对象从构造到发现问题为止的执行路径以及执行上下文。执行路径包括该对象的一系列方法调用，除了由它的其他方法进行的内部方法调用。执行上下文包括各方法调用所需的参数值。

例如，名称为 testObj 的对象在运行时具有如图 4 所示的交互。对象 testObj 是类 ClassTarget 的实例。它是通过调用构造器 ClassTarget(String) 创建的。在程序运行期间的某些时候，方法 methodA、methodB、methodC 和 methodD 被依次调用。当 methodD 被调用时发生问题。由于 methodC 是由 methodB 内部调用的，所以对象 testObj 的执行路径应当是 ClassTarget->methodA->methodB->methodD.

传递给方法调用的参数在单元测试用例中被当作应被复制或被模拟的测试数据。原始类型（例如，int、float 和 char）的值和可序列化（serializable）的对象（例如，图 4 中的字符串“test”）可以被复制，而其他不可序列化（即不可被复制）的对象（例如，图 4 中的 obj_b）可以被模拟。只有当一个对象的所有字段也是可序列化的或者是原始类型的时候，该对象才是可序列化的。复制物或模拟对象在运行时可提供与原始的被复制物或被模拟对象相同的行为。

当修改后的该程序执行时或执行后，系统可生成如下的测试用例。

```
String s1 = “test”;
String s2 = “aaa”;
DataObject o1 = new Mock_DataObject();

ClassTarget testObj = new ClassTarget(s1);
testObj.methodA(s2);
testObj.methodB(o2);
testOjb.methodD(); // 发现问题
```

图 5 示出了根据本发明的一个实施例的上述用于生成可再现运行时问题的测试用例的方法中的根据问题列表以及执行路径和执行上下文生成单元测试用例的步骤 104 中的子步骤。

如图所示，在步骤 501，遍历问题列表中所有非期望异常或预定行为规则的违反。

在步骤 502，对于所遍历的每一项非期望异常或预定行为规则的违反，获得所述捕获代码所记录的相应的执行路径和执行上下文以及所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值。

在步骤 503，根据所访问的所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值生成所述不可被复制的对象的模拟对象。优选地，如本领域的技术人员可理解的，可使用以下方法来生成模拟类。生成被模拟对象的子类，作为模拟该对象的模拟类。该子类中的模拟方法将覆盖被模拟对象中的原始方法并返回与原始方法的返回值相同的值。这说是说，该子类的实例对象具有与原始对象相同的行为。如本域的技术人员可理解的，还可通过上面提到的插入字节码的方法动态实现模拟。

在步骤 504，根据所述所有相关的执行路径及执行上下文以及所生成的模块类生成测试数据结构和测试目标执行路径，并将它们组合成单元测

试用例。

所生成的单元测试用例可被提供给开发团队以便利于对在测试执行中所发现的问题进行调试，或者便利于在程序被修改后进行的回归测试。

以上主要以 Java 面向对象程序为例描述了本发明的用于自动生成可再现运行时问题的单元测试用例的方法的实施例，应指出的是，本发明也适用于其他的面向对象程序。而且因为本发明的核心思想并不依赖面向对象，所以本发明也适用于非面向对象的程序。比如，在非面向对象的程序中，可以以单个函数作为进行单元测试的程序单元。这样，根据本发明的捕获代码和问题检测代码将用于记录所确定函数的执行路径和执行上下文，以及检测在其执行过程中所可能发生的所确定的非期望异常或预定规则的违反，并据此生成单元测试用例。

以上描述的根据本发明的实施例的用于自动生成可再现运行时问题的单元测试用例的方法主要是根据在验证测试之前确定的所关注的目标类及运行时问题修改目标程序，然后通过在验证测试过程中执行该修改后的目标程序来自动生成可再现运行时问题的单元测试用例的。根据本发明的用于自动生成可再现运行时问题的单元测试用例的方法另一个实施例，还可以不是在验证测试之前确定所关注的目标类和运行时问题并修改程序，而是首先进行正常的验证测试并产生缺陷描述；或者可以在根据本发明的上述实施例的方法中的测试执行步骤中除自动生成可再现所确定的运行时问题的单元测试用例之外，生成关于其他未确定的运行时问题的缺陷描述。开发者在得到所述缺陷描述后经过简单调试确定缺陷发生的代码位置，然后通过自定义新的规则或者是直接在代码中插入能够触发本发明生成测试用例的异常代码，来直接得到测试用例。这些测试用例可以被加入到自动化的单元测试过程中，避免手工开发单元测试用例，来降低验证新代码变更的代价。

图 6 示出了根据本发明的该另一个实施例的用于自动生成可再现运行时问题的计算机程序单元测试用例的方法的步骤。如图所示，在步骤 601，根据该计算机程序的验证测试后生成的测试缺陷描述，确定缺陷的性质及

发生缺陷的程序单元。在步骤 602，在所述程序中加入捕获代码和问题检测代码，所述捕获代码被配置为用于记录该程序中的所述程序单元的执行路径及执行上下文，而所述问题检测代码根据所确定的缺陷的性质被配置为用于检测该程序中所述程序单元的执行中的所述缺陷。在步骤 603，测试执行该程序。最后在步骤 604，根据检测到的所述缺陷以及所述捕获代码记录的相关的执行路径及上下文生成单元测试用例。

根据本发明的另一个方面，还提供了一种用于对计算机程序进行调试的方法。优选地，该方法包括以下步骤：使用前述用于自动生成可再现运行时问题的单元测试用例的方法在验证测试环境中自动生成该计算机程序的单元测试用例；以及通过在开发环境中运行所述单元测试用例来进行调试。由于所述单元测试用例是在验证测试期间针对开发者预先确定的目标类和可能发生的运行时问题自动生成的，所以在开发环境中运行所述单元测试用例将再现在测试环境中所遇到的运行时问题，从而便利了程序的调试和问题的解决。

根据本发明的又一个方面，还提供了一种用于对计算机程序进行回归测试的方法。优选地，该方法包括以下步骤：使用前述用于自动生成单元测试用例的方法生成该计算机程序的单元测试用例；以及在该计算机程序被修改后，通过在所述修改的计算机程序上运行所述单元测试用例来对该计算机程序进行回归测试。使用这种方法，可以自动生成可用于回归测试的单元测试用例，避免了开发者进行繁琐的手工开发。所自动生成的单元测试用例既可由开发者在对程序进行修改或升级后用于检查旧的已被解决的问题是否依然是被解决的，也可由测试者用于进行相同的检查，或由开发者和测试者双方使用。

根据本发明的再一个方面，还提供了一种计算机程序测试方法。优选地，该方法包括以下步骤：在所述程序中加入捕获代码和问题检测代码，其中所述捕获代码被配置为用于记录该程序中所关注的目标程序单元的执行路径及执行上下文，而所述问题检测代码被配置为用于检测并记录该程序中所述所关注的程序单元的执行所可能抛出的所关注的非期望异常以及

所可能产生的所关注的预定行为规则的违反；以及测试执行经过所述加入的该程序，以检测并记录所述所关注的非期望异常以及预定行为规则的违反。优选地，该方法还包括定义规则的步骤，以便定义所述所关注的预定行为规则。一种定义规则的方法是通过为每一条规则确定规则模板、目标方法和开发检验代码。通过这种方法，有可能在测试过程中发现尚未表现出的潜在的缺陷，以有助于程序测试。

以上描述了根据本发明的实施例的用于自动生成可再现运行时问题的单元测试用例的方法、以及基于该方法的用于对计算机程序进行调试的方法、用于对计算机程序进行回归测试的方法、以及计算机程序的测试方法。相应地，本发明还提供了一种用于自动生成可再现运行时问题的计算机程序单元测试用例的系统。

图 7 示出了根据本发明的实施例的用于自动生成可再现运行时问题的计算机程序单元测试用例的系统 700 的框图。如图所示，该系统 700 包括：修改模块 703，该修改模块用于根据计算机程序中所关注的一个或多个目标程序单元和运行时问题修改该程序；测试执行模块 704，该测试执行用于测试执行所述被修改后的程序；以及单元测试用例生成模块 705，该单元测试用例生成模块用于根据在所述所关注的程序单元的执行中出现的所述所述所关注的运行时问题自动生成单元测试用例。

优选地，所述运行时问题包括非期望的异常和预定行为规则的违反。

优选地，所述修改模块 703 被配置为用于向该程序加入捕获代码，所述捕获代码被配置为用于记录该程序中所述所关注的程序单元的执行路径和执行上下文；以及向该程序加入问题检测代码，所述问题检测代码被配置为用于检测所述程序单元的执行所可能抛出的所关注的非期望异常以及所可能产生的所关注的预定行为规则的违反。

优选地，所述问题检测代码还被配置为用于每当检测到所述非期望异常或预定行为规则的违反时，就触发所述单元测试用例生成模块 705 根据所述捕获代码所记录的执行路径及执行上下文生成单元测试用例。

作为另一种选择，所述问题检测代码还被配置为用于每当检测到所述

非期望异常或预定行为规则的违反时，将所述非期望异常或预定行为规则的违反记入问题列表，并且所述单元测试用例生成模块 705 被配置为用于当所述测试执行模块进行的测试执行过程结束后，根据所述问题列表以及所述捕获代码所记录的执行路径及执行上下文生成单元测试用例。

优选地，所述计算机程序为面向对象的程序，更具体地例如为 Java 字节码程序，所述程序单元为类，所述执行路径为所述目标类的各实例对象的每一次外部方法调用，并且所述执行上下文为所述目标类的各实例对象的每一次外部方法调用的参数的值。但应指出，本发明也适用于非面向对象的程序。比如，在非面向对象的程序中，可以以单个函数作为进行单元测试的程序单元。这样，根据本发明的捕获代码和问题检测代码将用于记录确定函数的执行路径和执行上下文，以及检测在其执行过程中所可能发生的所确定的非期望异常或预定规则的违反，并据此生成单元测试用例。

优选地，所述捕获代码进一步被配置为用于记录所述目标类的各实例对象的调用参数中不可被复制的对象的被调用方法、调用参数及其返回值，并且所述单元测试用例自动生成模块 705 被配置为用于：根据所述捕获代码所记录的所述目标类的各实例对象的调用参数中不可被复制的对象的被调用方法、调用参数及其返回值生成所述不可被复制的对象的模拟对象；以及根据所述捕获代码所记录的执行路径及执行上下文以及所述生成的模拟对象生成单元测试用例。

修改模块 703 的输入为要对其生成测试用例的目标程序以及用户所确定的所关注的目标类和运行时问题，其输出为经过修改的包含所述捕获代码和问题检测代码的目标程序。该经过修改的目标程序将被提供给测试执行模块 704 进行测试执行。

在本发明的一优选实施例中，所述修改模块 703 是字节码插入工具例如普通的字节码插入工具，其通过修改目标程序的类文件来加入所述捕获代码和问题检测代码。但是，在本发明的其他实施例中，还提供了可以在源代码中被调用的方法。这样，修改模块 703 可以为用于在源代码中加入所述捕获代码和问题检测代码的工具。或者，可以人工加入所述捕获代码

和问题检测代码以触发测试用例的生成，并且在这种情况下，修改模块 703 可被理解为进行人工加入时的编辑工具。

在本发明的一优选实施例中，所述测试执行模块 704 为普通的测试系统或其部分。在本发明的其他实施例中，该测试执行模块 704 为专用于本发明的测试模块。此外，测试执行模块 704 与所述单元测试用例生成模块 705 既可以本领域的技术人员可知的多种方式，例如以插件的方式或者以包含这两种功能的单个模块的方式，集成在一起；也可以作为两个在物理上连接在一起的分立的模块；也可以作为两个仅仅在逻辑在连接在一起、在物理上相互分离和独立的模块，并且在这种情况下两个模块仅仅是依靠测试执行模块 704 所输出的、作为单元测试用例生成模块 705 的输入的执行路径和执行上下文记录以及运行时问题列表形成逻辑上的连接关系。

单元测试用例生成模块 705 的输入为测试执行模块 704 所输出的执行路径和执行上下文记录以及发现的问题，其输出为所生成的单元测试用例。所述单元测试用例可被提供给开发团队以便利于对在测试执行中所发现的问题进行调试，或者便利于在程序被修改后进行的回归测试。优选地，所述单元测试用例生成模块 705 被配置为用于将单元测试用例形成为包含 JUnit 的测试用例类的 Java 源文件。

优选地，该系统 700 还包括确定模块 702，该确定模块用于确定所述程序中所述所关注的目标程序单元以及所述所关注的非期望异常和/或预定行为规则的违反。

确定模块 702 的输出是包含目标类和运行时问题的配置信息。确定模块 702 可以是一 GUI 工具。该工具除了让用户确定关注的目标类以及异常之外，还会列出所有已定义的规则树，让用户勾选需要应用的规则。当然，GUI 工具实际上只是提供了一个方便的手段来生成这些提供给第二个步骤的配置信息，用户完全可以手工编写这些配置，用于只有命令行可供使用的场合。此外，确定模块 702 的功能也可以隐含在修改模块 703 的操作中，即在通过工具或手工修改目标程序的过程中在确定在何处修改及如何修改目标程序时体现所关注的目标类和运行时问题，并且在这种情况下，

该系统 700 将没有确定模块 702。

优选地，该系统 700 还包括规则定义模块 701，该规则定义模块用于定义可由所述确定模块 702 确定的预定行为规则。预定行为规则描述了所确定程序单元在运行中所应当表现出的行为，包括正确的方法调用顺序、有效的方法参数和返回值等。定义好的规则可被存储在一规则库中以便由确定模块 702 从中检索，也可以被直接提供给确定模块使用。可以针对每一待测试的程序或一程序的每一次测试生成一组行为规则，也可以创建由多个程序共享的行为规则库。

该规则定义模块 701 优选地被配置为用于通过以下步骤来定义规则：确定包含目标方法和检验代码的占位符的规则模板；确定目标方法；以及开发检验代码。规则模板是指预先定义好的一些模式，例如“Never call <X> when <CONDITION>”, “Never call <X> before <Y> of an object when <CONDITION>”等等。目标方法即是定义模板中的“<X>”、“<Y>”具体是什么类的什么方法，这决定了检验代码插入的位置。检验代码即是在目标方法被调用的前后对方法的对象、参数、返回值等等实现模板中的“<CONDITION>”检验。在本发明的一个优选实施例中，检验代码通过继承一个规则抽象类来实现，如对于上述模板“Never call <X> when <CONDITION>”需要继承 AbstractNeverCallXRule 类，并重载其中的 matchedX 方法即可针对取得的上下文进行检验。规则的抽象类是模板的一部分，通过它隐藏了模板的具体实现，从而开发规则时可以只专注于检验代码的开发。

应指出，通过规则模板以及规则的抽象类来定义待确定的规则的方法只是本发明的定义和确定规则的一种实现方式，本发明也可采用其他定义和确定规则的方式，例如通过直接插入规则的检验代码（定义和确定规则的步骤隐含在插入检验代码的过程中）等等。

优选地，所述修改模块 703、确定模块 702、和规则定义模块 701 位于开发环境中，而所述测试执行模块 704、和单元测试用例生成模块 705 位于测试环境中。但其他安置方式也是可能的，例如修改模块 703、确定模

块 702、规则定义模块 701 以及测试执行模块 704、和单元测试用例生成模块 705 均位于开发环境中，或均位于测试环境中，或同时位于两个环境中，或位于其他环境中。

图 8 示出了根据本发明的进一步的实施例的单元测试用例生成模块 705 的结构示意图。如图所示，单元测试用例生成模块 705 包括：问题列表访问模块 801，用于遍历由测试执行模块 704 所生成的问题列表中的每一项非期望异常或预定行为规则的违反；执行路径及上下文访问模块 802，用于针对所遍历的每一项非期望异常或预定行为规则的违反，访问所述捕获代码所记录的相应的执行路径和执行上下文以及所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值；模拟对象构造模块 803，用于根据所访问的所述不可被复制的对象在目标类中的被调用方法、调用参数及其返回值生成调用参数中的不可被复制的对象的模拟对象；以及单元测试用例构造模块 804，用于根据所访问的执行路径及执行上下文以及所生成的模拟对象构造单元测试用例。

以上描述的本发明的实施例的用于自动生成可再现运行时问题的单元测试用例的系统 700 主要用于这样的场景，即根据在验证测试之前确定的所关注的目标类及运行时问题修改目标程序，然后通过在验证测试过程中执行该修改后的目标程序来自动生成可再现运行时问题的单元测试用例。然而，根据本发明的实施例的该用于自动生成可再现运行时问题的单元测试用例的系统 700 还可以用于这样的场景，即不是在验证测试之前确定所关注的目标类和运行时问题并修改程序，而是首先进行正常的验证测试并产生缺陷描述；或者可以在上述在验证测试之前确定所关注的目标类及运行时问题的场景中，在测试执行步骤中除自动生成可再现所确定的运行时问题的单元测试用例之外，还生成关于其他未确定的运行时问题的缺陷描述。在这样的场景中，开发者在得到所述缺陷描述后经过简单调试确定缺陷发生的代码位置，然后通过自定义新的规则或者是直接在代码中插入能够触发本发明生成测试用例的异常代码，来直接得到测试用例。这些测试用例可以被加入到自动化的单元测试过程中，避免手工开发单元测试用例，

来降低验证新代码变更的代价。

当根据本发明的实施例的用于自动生成可再现运行时问题的计算机程序单元测试用例的系统用于这样的场景时，所述规则模块 701 所定义的、和所述确定模块 702 所确定的规则是从在验证测试后生成的测试缺陷描述中得出的，所述确定模块 702 所确定的所关注的异常和目标类也是从测试缺陷描述中得出的。然后，在经修改模块 703 根据所述得出的目标类以及异常和/或规则修改了目标程序之后，优选地在开发环境中的测试执行模块 704 中进行测试运行，并由单元测试用例生成模块 705 根据在测试执行中所生成的执行路径和执行上下文以及运行时问题生成单元测试用例。该单元测试用例将主要用于未来对该目标程序进行的回归测试。

根据本发明的另一个方面，还提供了一种用于对计算机程序进行调试的系统。优选地，该系统包括：前述任何一种用于自动生成可再现运行时问题的单元测试用例的系统 700；以及调试装置，用于通过运行所述用于自动生成可再现运行时问题的计算机程序单元测试用例的系统 700 所生成的单元测试用例来对计算机程序进行调试。由于所述单元测试用例是在验证测试期间针对开发者预先确定的目标类和可能发生的运行时问题自动生成的，所以在开发环境中运行所述单元测试用例将再现在测试环境中所遇到的运行时问题，从而便利了程序的调试和问题的解决。

根据本发明的又一个方面，还提供了一种用于对计算机程序进行回归测试的系统。优选地，该系统包括：前述任何一种用于自动生成可再现运行时问题的单元测试用例的系统 700；以及回归测试装置，用于在该计算机程序被修改后，通过在所述修改的计算机程序上运行所述用于自动生成可再现运行时问题的计算机程序单元测试用例的系统 700 在该计算机程序被修改之前所生成的单元测试用例来对该计算机程序进行回归测试。使用这种方法，可以自动生成可用于回归测试的单元测试用例，避免了开发者进行繁琐的手工开发。所自动生成的单元测试用例既可由开发者在对程序进行修改或升级后用于检查旧的已被解决的问题是否依然是被解决的，也可由测试者用于进行相同的检查，或由开发者和测试者双方使用。

根据本发明的另外一个方面，还提供了一种计算机程序测试装置。优选地，该装置包括：修改模块，用于在所述程序中加入捕获代码和问题检测代码，所述捕获代码被配置为用于记录该程序中所关注的目标程序单元的执行路径及执行上下文，而所述问题检测代码被配置为用于检测并记录该程序中所述所关注的程序单元的执行所可能抛出的所关注的非期望异常以及所可能产生的所关注的预定行为规则的违反；以及测试执行模块，用于测试执行经过所述加入的该程序，以检测并记录所关注的目标程序单元的执行中出现的所述所关注的非期望异常以及预定行为规则的违反。优选地，该装置还包括规则定义模块，用于定义所述所关注的预定行为规则。优选地，该规则定义模块被配置为用于通过为每一条规则确定规则模板、目标方法以及开发检验代码来定义规则。使用这种计算机程序测试装置，有可能发现尚未表现出的潜在的缺陷，以有助于程序测试。

以上参照附图描述了根据本发明的实施例的用于自动生成可再现运行时问题的测试用例的方法和系统，用于对计算机程序进行调试的方法和系统，用于对计算机程序进行回归测试的方法和系统，以及计算机程序的测试方法和系统。

应理解，附图仅作说明之用，而不构成对本发明的限制。附图可以但不一定示出本发明的某单个实施例，而可以示出多个实施例的组合。此外，可以考虑用上文中所述的特征和元素的任意组合来实施和实践本发明，而无论它们是否涉及不同的实施例。相关领域内的技术人员可认识到，可实现本发明而没有特定实施例的一个或多个特定特征或优点。在另外的情况下，可在一些实施例内实现另外的特征和优点，而它们可能不存在于本发明的任何实施例内。涉及特征和优点的语言应被理解为意指与实施例相联系地描述的特定特征、优点和特性被包含在本发明的至少一个实施例内。因此，此整个说明书内的对特征和优点的讨论以及类似语言可以但不必须指的是相同实施例。

另外，在各实施例中，本发明提供了优于现有技术的大量优点。然而，尽管本发明的实施例可获得优于其他可能的解决方案和/或优于现有技术

的优点，某一具体优点是否由给定的实施例获得并不构成对本发明的限定。因此，下面的方面、特征、实施例和优点仅作说明之用而不应被看作是所附权利要求的要素或限定，除非权利要求中明确提出。类似地，谈到“本发明”不应被解释为对此处所披露的任何发明主题的概括，也不应被看作是所附权利要求的要素或限定，除非权利要求中明确提出。

本发明的这些特征和优点可从以上说明以及所附权利要求内完全清楚地看到，或者可通过如上文所述地实践本发明来了解。

此说明书内所述的许多功能单元已被标记为模块，以便更特别地强调它们的实现独立性。例如，模块可实现为包括定制 VLSI 电路或门阵列、现成的半导体例如逻辑芯片、晶体管或其它离散元件的硬件电路。模块也可实现为可编程硬件设备例如现场可编程门阵列、可编程阵列逻辑、可编程逻辑设备等。

模块还可实现为可被各种处理器执行的软件。可执行代码的被标识的模块可包括例如可被组织成对象、过程或函数的计算机指令的一个或多个物理或逻辑块。但是，被标识的模块的可执行代码不必在物理上位于一起，而是可包括存储在不同位置的不同指令，这些不同指令当在逻辑上连接在一起时构成该模块并实现模块的规定目的。

实际上，可执行代码的模块可以是单个指令或许多指令，甚至可分布在一些不同的代码段上、不同程序中以及几个存储设备上。类似地，操作数据在这里可在模块内被标识和说明，并且可体现为任何合适的形式且组织成任何合适类型的数据结构。操作数据可被聚集成单个数据集，或者可分布在不同位置上包括在不同存储设备上，并且可至少部分地仅仅作为系统或网络上的电子信号存在。

此外，在上面的说明内，提供了许多特定细节例如编程、方法步骤、用户选择、系统模块等的示例，以便提供对本发明的实施例的透彻理解。但是，本领域内的技术人员可认识到本发明可实现为不具有一个或多个所述特定细节，或者可实现为具有其它步骤、部件。在另外的实施例内，没有详细示出或说明公知的部件或操作以避免掩盖本发明的方面。

另外，上文中任何具体的模块、装置、系统、方法、步骤的命名仅是为了方便而使用的，故而不构成对本发明的限制。

一般地给出上文的示例性流程图，作为逻辑流程图。因而，所示的顺序和标记的操作指示所给出的方法的实施例。可设想在功能、逻辑或效果上与所示方法的一个或多个操作或其部分等效的其它的操作和方法。另外，特定方法发生的顺序可能或可能不严格遵循所示的对应操作的顺序。某些步骤可以合并在一起，或进一步划分了更细的步骤，或添加一些新步骤，或去除一些步骤。

类似地，所描述的系统和装置的模块之间的连接关系可以改变，它们可以合并成更大的模块，或被进一步地划分为更小的模块，也可以添加某些新的模块，或去除某些模块，只要它们能够实现根据本发明的功能，这些改变都处于本发明的精神和范围之内。

本发明可以硬件、软件、或硬件与软件的结合的方式实现。本发明可以集中的方式在一个计算机系统中实现，或以分布方式实现，在这种分布方式中，不同的部件分布在若干互连的计算机系统中。适于执行本文中描述的各方法的任何计算机系统或其它装置都是合适的。一种典型的硬件和软件的组合可以是带有计算机程序的通用计算机系统，当该计算机程序被加载和执行时，控制该计算机系统而使其执行本文中描述的方式。

本发明也可体现在与计算机系统一起使用的程序产品中。该程序产品包含使能实现本文中描述的方法的所有特征，并且当其被加载到计算机系统中时，能够执行这些方法。该程序产品可被包含在多种信号承载介质中。典型的信号承载介质包括但不限于：（i）永久存储在不可写存储介质（例如计算机中的只读存储器装置，例如可由 CD-ROM 驱动器读取的 CD-ROM 盘）上的信息；（ii）存储在可写存储介质（例如软盘驱动器中的软盘或磁盘驱动器）上的可更改的信息；以及（iii）通过例如包括无线通信在内的计算机或电话网络等通信介质传送给计算机的信息。当这种信号承载介质携带指导本发明功能的、计算机可读的指令时，其代表本发明的实施例。

尽管已参照优选实施例具体示出和说明了本发明，但是本领域内的那些技术人员应理解，可在形式和细节上对其进行各种改变而不会背离本发明的精神和范围，本发明的范围由所附权利要求限定。

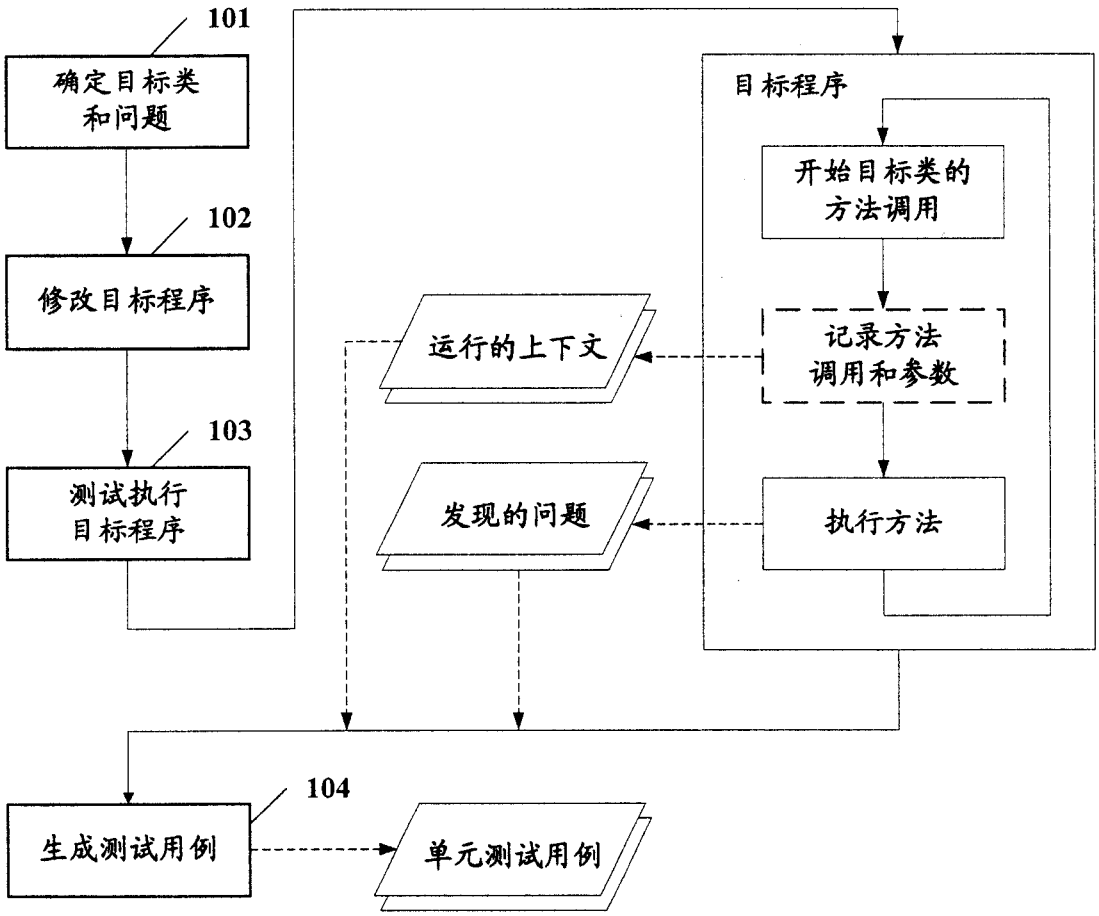

101
确定目标类和问题
102
修改目标程序
103
测试执行目标程序
104
生成测试用例
单元测试用例
运行的上下文
发现的问题
目标程序
开始目标类的方法调用
记录方法调用和参数
执行方法

图1

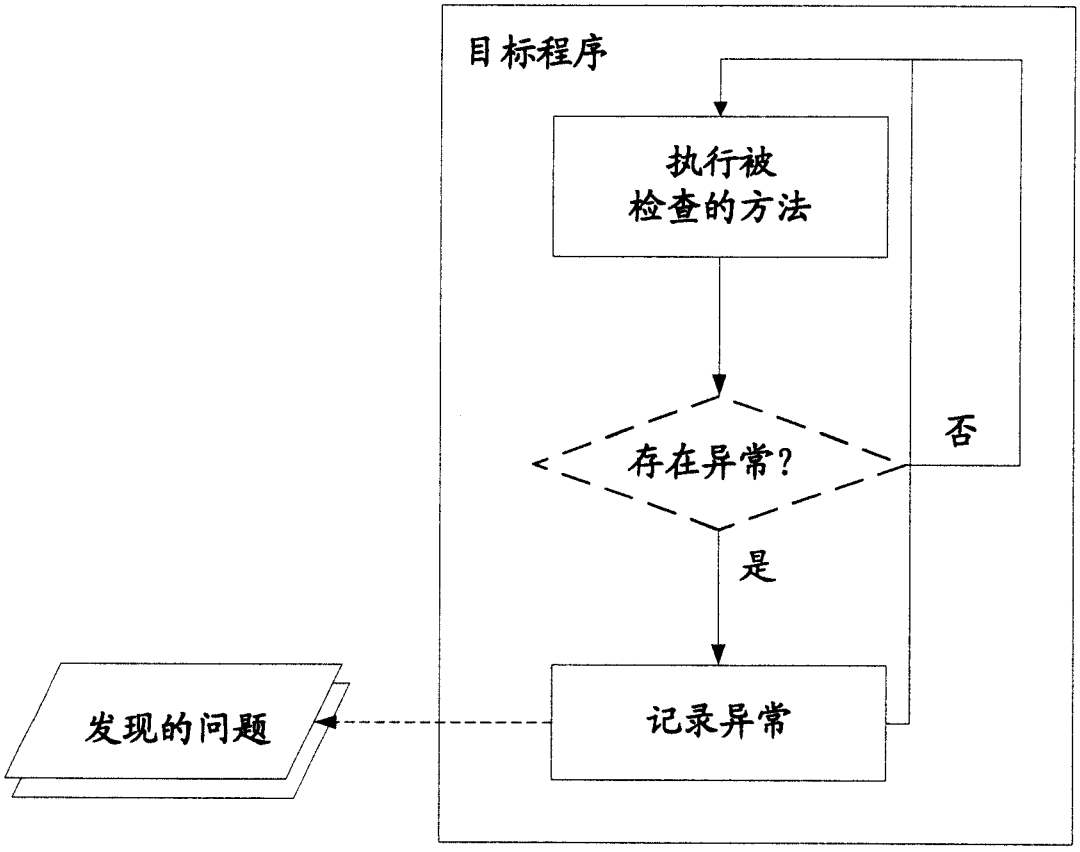


图2

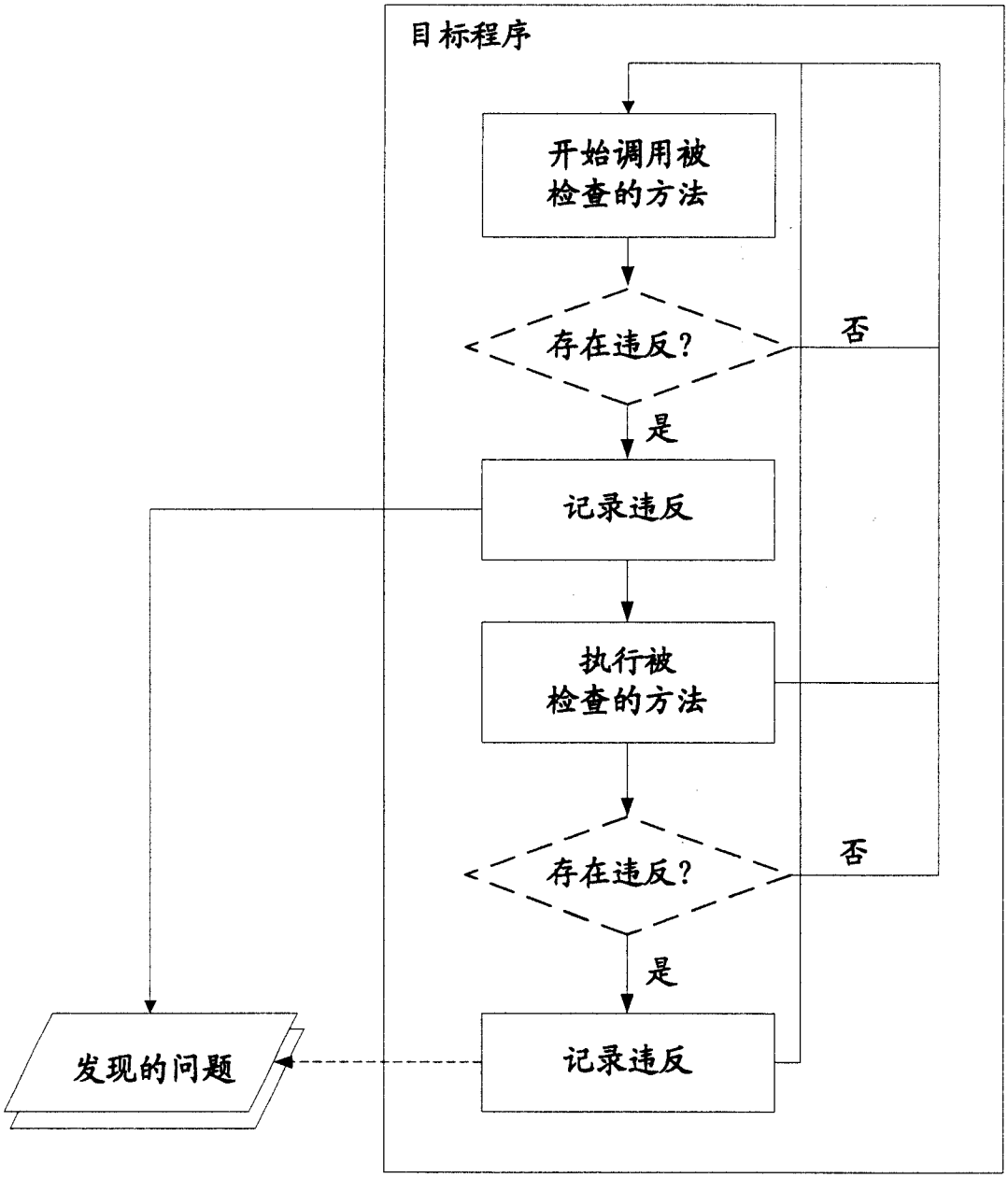


图3

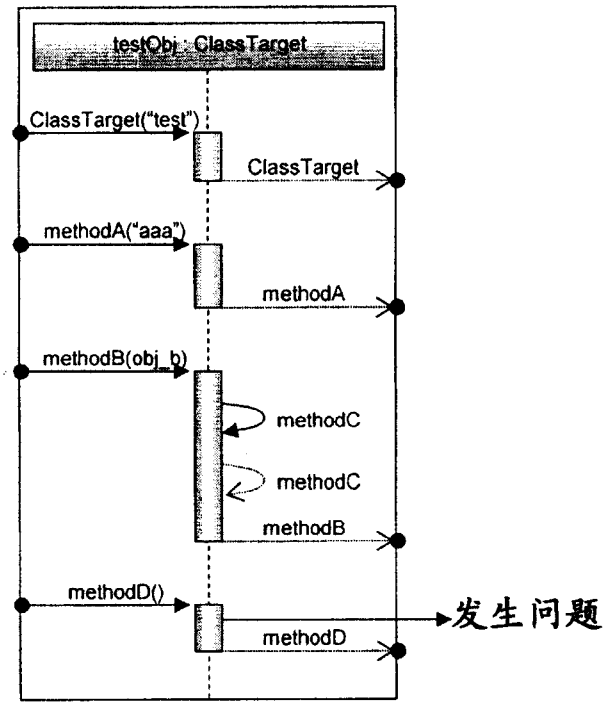

testObj : ClassTarget
ClassTarget("test")
ClassTarget
methodA("aaa")
methodA
methodB(obj_b)
methodC
methodC
methodB
methodD()
发生问题
methodD

图 4

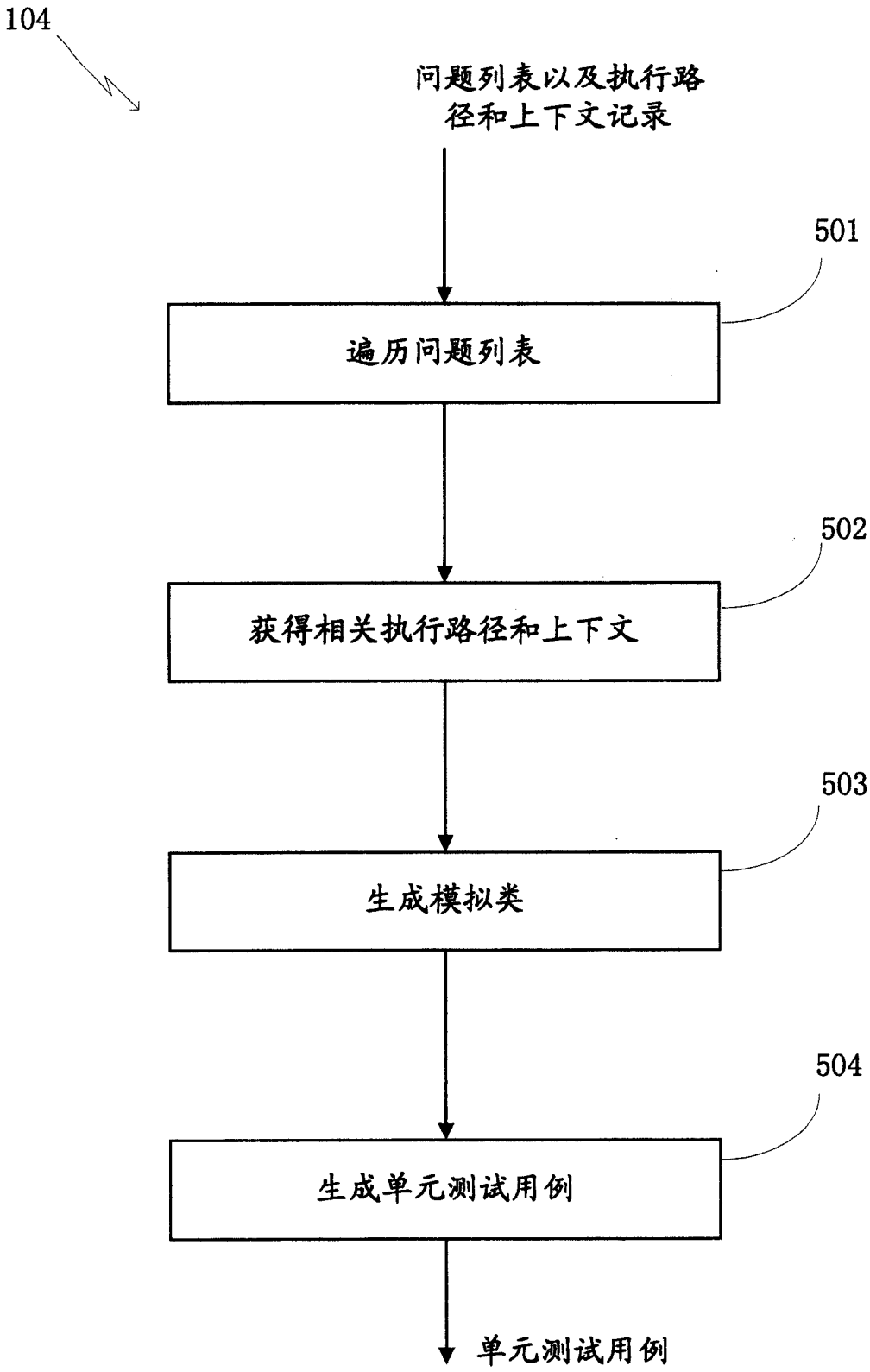

104
问题列表以及执行路径和上下文记录
501
遍历问题列表
502
获得相关执行路径和上下文
503
生成模拟类
504
生成单元测试用例
单元测试用例

图5

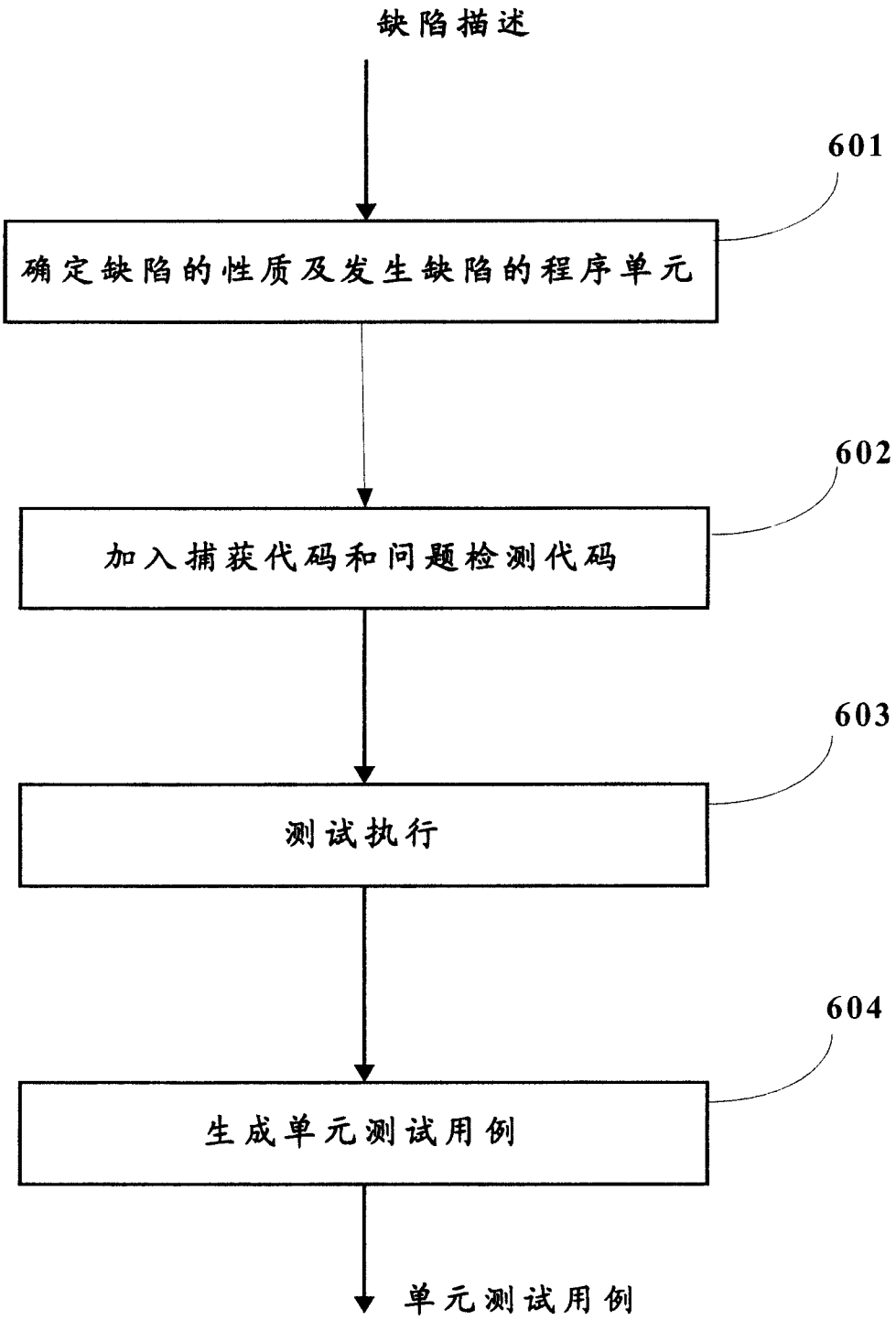

缺陷描述
601
确定缺陷的性质及发生缺陷的程序单元
602
加入捕获代码和问题检测代码
603
测试执行
604
生成单元测试用例
单元测试用例

图 6

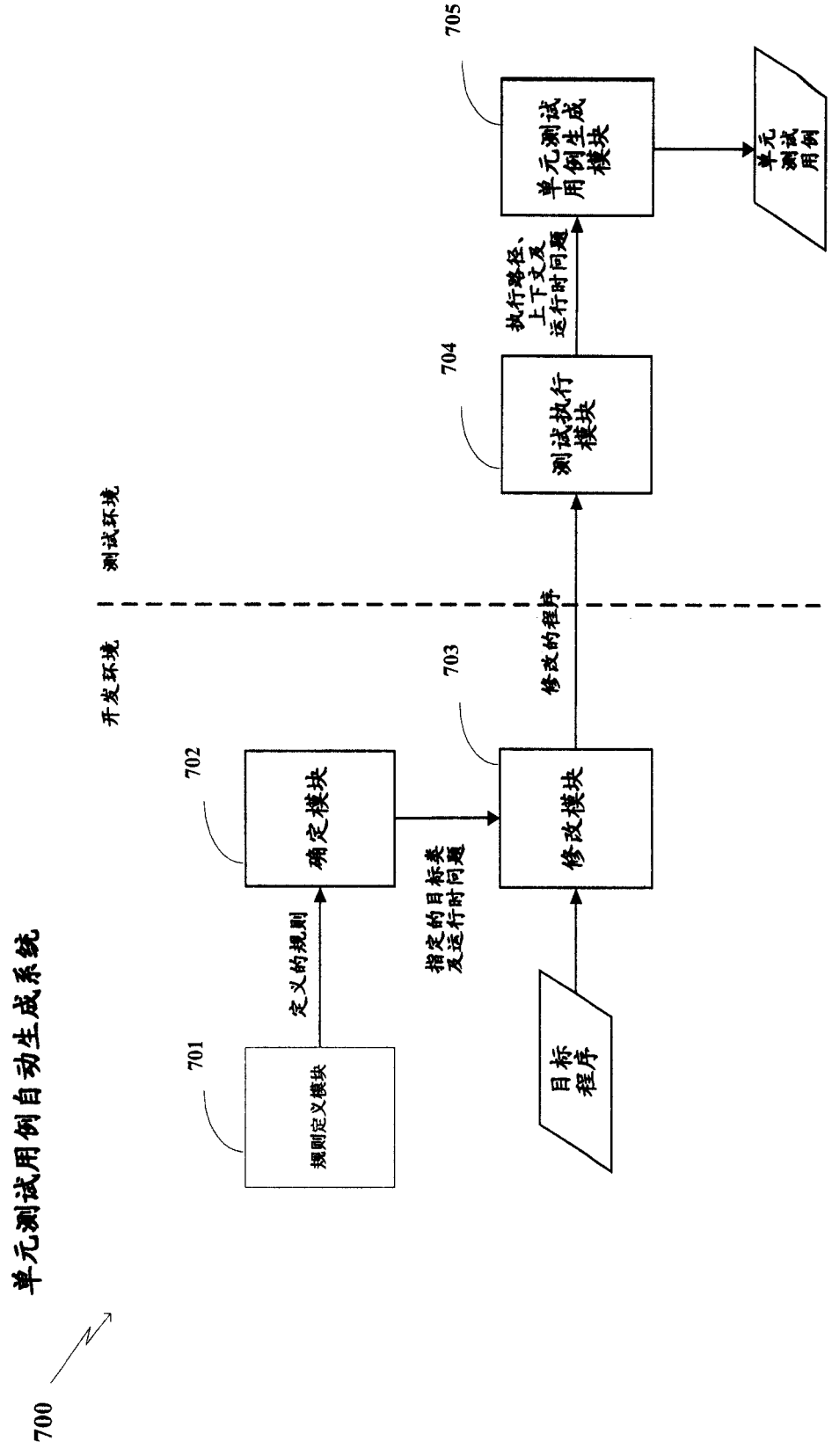

单元测试用例自动生成系统
700
开发环境
测试环境
701
规则定义模块
定义的规则
702
确定模块
指定的目标类
及运行时问题
703
修改模块
目标
程序
修改的程序
704
测试执行
模块
执行路径、
上下文及
运行时问题
705
单元测试
用例生成
模块
单元
测试
用例

图7

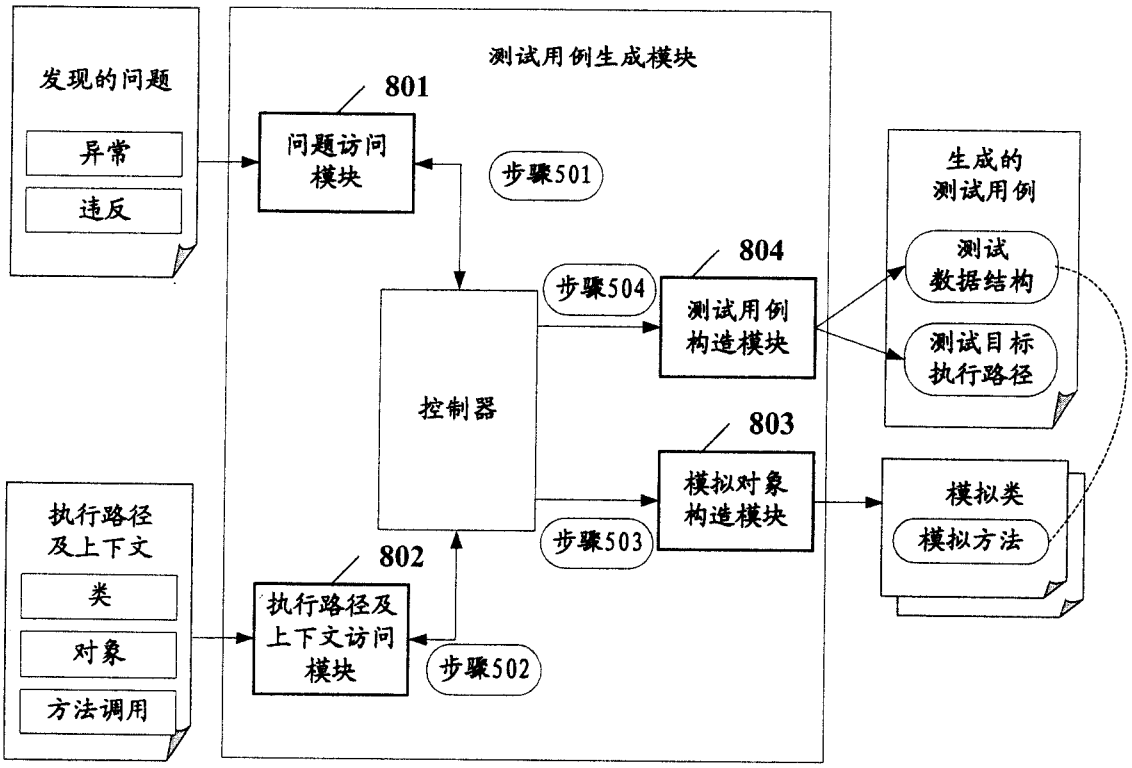

发现的问题
异常
违反
测试用例生成模块
801
问题访问模块
步骤501
控制器
步骤504
804
测试用例构造模块
生成的测试用例
测试数据结构
测试目标执行路径
803
模拟对象构造模块
步骤503
模拟类
模拟方法
执行路径及上下文
类
对象
方法调用
802
执行路径及上下文访问模块
步骤502

图8