文章编号:1671-4598(2025)01-0029-07 DOI:10.16526/j.cnki.11-4762/tp.2025.01.004 中图分类号:TP315.69 文献标识码:A

基于 RTC 闭环覆盖率驱动的自动化测试系统研究

李 智,刘 欣,孙 肖,王月波,李海霞
(中国电子科技集团 第 10 研究所,成都 610000)

摘要:为解决手工测试、自动化测试过程中漏测问题,优化回归测试,提高测试精准度和执行效率;首先,采用头脑风暴和项目历史经验的方法,分析了软件测试过程中产生漏测的原因,列举了常用解决漏测问题的措施;然后,常用方法只能反应软件开发过程中某一方面会引起软件质量问题,不能在软件全生命周期综合定量反馈软件质量,针对该问题,从软件工程 RAD 开发模型理论引入需求覆盖、测试用例覆盖、代码覆盖测量指标,分析 3 个指标之间关联关系,提出“软件需求-代码-测试用例”闭环覆盖率方法监测软件测试质量;其次,设计了基于软件需求、代码、测试用例闭环率驱动的自动化测试系统模型,并阐述了其设计模型和实现方法,同时开发了原型系统;最后,进行案例实验,验证方法可行性和效果,并针对实验过程的问题提出后续改进方向。

关键词:漏测;覆盖率;软件自动化测试;RAD;RTC;测试质量

Research of Automated Testing System Driven with RTC Closed Loop Coverage

LI Zhi, LIU Xin, SUN Xiao, WANG Yuebo, LI Haixia
(The 10th Research Institute of CETC, Chengdu 610000, China)

Abstract: To solve missed testing issues in manual and automated testing processes, optimize regression testing, and improve testing accuracy and execution efficiency; Firstly, adopt the methods for brainstorming and project history experience to analyze the reasons for missed testing during software testing process and common measures to solve missed testing; Secondly, common methods can only reflect that a certain aspect of the software development process will cause software quality problems, and cannot provide a comprehensive quantitative feedback on software quality during entire software life cycle; Then, to address this issue, introduce the measurement indicators of requirements coverage, test case coverage, and code coverage from the development model of the software engineering RAD, analyze the correlation between the three indicators, and propose a closed-loop coverage method of “software requirements code test cases” to monitor software testing quality; And then, design an automated testing system model driven by software requirements, code, and test case closed-loop rate, and elaborate its design model and implementation method. At the same time, develop a prototype system; Finally, verify the feasibility and effectiveness of the method on case experiments, and provide further improvement directions for the problems encountered in the experiment.

Keywords: missing measurement; coverage rate; software automation testing; RAD; RTC; test quality

0 引言

软件测试是软件开发生命周期中的一个重要环节,是保证软件质量的关键。在常见的测试方法中,手工测试耗时长且易漏测,且随着软件规模、复杂度的不断提高,手工测试的效率无法满足快速迭代,且质量越来越难以保证。随着自动化测试的出现,提高了软件测试效率,但漏测的问题仍然未得到有效解决,如何通过定量的方法评估测试的完整性,并结合到自动化测试系统中,提高测试效率的同时,又可监测自动化测试的完整

收稿日期:2023-11-12; **修回日期**:2024-01-05。
作者简介:李 智(1983-),男,研究生,高级工程师。
引用格式:李 智,刘 欣,孙 肖,等.基于 RTC 闭环覆盖率驱动的自动化测试系统研究[J].计算机测量与控制,2025,33(1):29-35.

性和质量。首先从可能导致漏测的原因开始分析，并结合软件工程开发模型分析漏测在代码、需求、测试用例的底层联系逻辑，提出“软件需求－代码－测试用例”闭环追踪方法，通过理论体系牵引，研究基于软件需求、代码、测试用例闭环覆盖率统计的自动化测试系统，以及其实现的方法，同时集成 TestLink、Selenium、TestNg 和 Jacoco 等工具，通过二次开发实现该自动化测试原型系统，并进行实际案例实验，验证方法可行性和效果。

1 软件漏测分析

1.1 软件漏测的原因

通过历史型号项目出现的软件漏测情况，结合头脑风暴，分析软件漏测原因可能为：

1）需求不完整或不准确。软件需求规格说明对软件功能和性能等描述不完整、不充分和准确等，导致测试人员未充分设计测试用例验证软件功能、性能，从而漏测；

2）测试用例设计不完整和充分。测试用例未完全覆盖软件需求规格说明，或测试人员未采用功能分解、等价类划分、边界值、因果图等方法充分对软件功能需求、场景进行测试设计；

3）需求变更或设计变更但测试未迭代：在测试过程中，如果需求或设计发生了变更，而测试人员没有及时了解和更新测试用例，导致漏测；

4）测试环境与实际运行环境存在差异。测试环境与实际运行环境在操作系统、数据库、中间件、环境参数配置等存在差异，导致测试用例在测试环境和实际运行环境产生不同的执行结果，出现测试环境用例通过，但实际环境失败的情况；

5）测试执行人为疏忽。测试人员可能会因为疲劳、经验不足或不专注等原因，出现测试用例未完全执行，测试结果判断错误，导致漏测问题；

6）软件变更影响分析不足，回归测试覆盖不全。开发人员修复缺陷，引入新的缺陷，而测试人员对软件变更分析不充分，导致回归测试未全面验证变更受影响功能。

综合以上情况，软件漏测的原因既有人员经验、业务能力、职业素质等主观原因，也有运行环境差异、信息传递失误等客观原因。根据软件工程化阶段划分，漏测原因涉及了软件工程的需求分析阶段，设计阶段，测试阶段的测试设计和测试执行，通过归纳分析，本质是各阶段输出产品的充分性、准确性、一致性问题。

1.2 漏测防患措施

防患软件漏测的本质是提升各阶段产出的充分性，准确性，确保下一阶段的输入与上一阶段输出的一致性。常用措施是在各阶段增加人员加强监督、审查、评审，如：需求评审、设计评审、测试需求评审、测试环境有效性评估、测试执行监督等，降低个体犯错的概率，弥补个体专业技能不足。

表 1 软件工程各阶段产生漏测的因素

软件工程阶段		漏测原因	受影响文档或产品
需求分析阶段		需求不完整或不准确	需求规格说明
设计阶段		设计文档不完整或不准确	设计说明、用户手册
测试阶段	测试设计	需求变更或设计变更但测试未迭代；	测试用例
		测试用例设计不完整和充分；	测试用例
	测试执行	测试环境与实际运行环境存在差异	测试环境
		测试执行人为疏忽	测试执行记录
		软件变更影响分析不足，回归测试覆盖不全	测试用例

常规措施中增加人员监督、审核的方法虽然降低了犯错的概率，但同样存在人主观因素引入错误，因此，各阶段引入客观指标对产出物进行充分性、一致性评估成为有效且必要手段。常见的指标如下：

1）需求覆盖率：是软件需求覆盖用户需求、合同、建设方案等的比例，反映软件设计需求对用户实际需求的充分性和一致性；

2）测试用例覆盖率：是指测试用例对软件需求的覆盖程度，一般要求测试用例覆盖率达到 100%，反映测试设计的充分性，验证用户需求的一致性；

3）测试用例执行率：是指实际执行测试用例的比例，反映测试执行的充分性；

4）缺陷发现率：是指测试用例在执行过程中发现缺陷的比例，反映测试设计的有效性；

5）测试用例质量：测试用例质量是指测试用例的正确性、完整性和可靠性等；

6）测试投入充足率：测试投入充足率是指测试投入是否充分，包括测试时间、人力、资源等；

7）软件代码覆盖率：测试用例执行所覆盖的代码行数与代码总行数的比例，并分为语句覆盖、分支覆盖、条件覆盖等，反映测试执行的充分性。

2 RTC 闭环覆盖率方法论

常规漏测防患措施各指标只能单一反馈某一方面的质量，如需求覆盖率只能反映需求的正确实现，缺乏整体客观反映需求、测试充分性的指标体系。依据 RAD 模型，结合常规充分性评估指标，提出 RTC 闭环覆盖率（即“软件需求-测试用例-软件代码”闭环覆盖率），贯穿软件需求设计，软件代码实现，软件需求实现验证

等三大环节，整体反映软件测试质量。

2.1　RTC 闭环覆盖率描述

RTC 闭环覆盖率是指在软件需求、测试用例、软件代码形成的环形追踪关系中，软件需求与代码覆盖率 R、软件需求与测试用例覆盖率 T、代码测试覆盖率 C 所形成的闭环覆盖率，即 RTC Coverage。RTC 闭环覆盖率计算方式如下：

$$RTC = \frac{1}{3}R + \frac{1}{3}T + \frac{1}{3}C$$

RTC 为 100%时，代表软件需求、代码、测试等三大环节相互追溯达到 100%。RTC 值越高软件漏测率越低，其测试结果可行度越高。

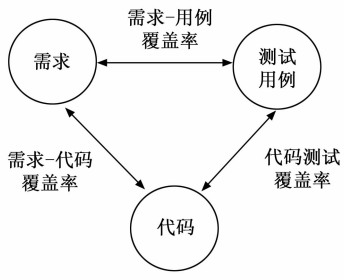

图 1　RTC 闭环覆盖率关系

2.2　理论依据

RAD（Rapid Application Development，快速应用开发）模型，也称为 V 模型，是软件开发与测试验证的重要工作模型，描述了软件开发、测试过程各阶段转换关系、依赖关系、输入输出产物，并核心阐述了确保软件开发各阶段正确性的对应测试验证级别。

V 模型左侧是软件开发过程依次经历的需求分析、软件设计、软件编码，上一阶段的产出作为下一阶段的输入，并通过下一阶段产物对上一阶段输出的 100%覆盖率，确保各设计阶段正确实现用户需求，即软件需求与软件代码的覆盖率 R。

V 模型右侧是软件测试依次开展的过程，即单元测试、部件测试、配置项测试、系统测试。在测试用例执行过程中被测软件中对应软件代码被调用运行，被执行代码占比，即为代码覆盖率。代码覆盖率不仅侧面反应了测试用例与代码追溯关系（即软件代码与测试用例覆盖率 C），也反映了测试的充分性，代码覆盖率越高表面测试执行越充分。

在 V 字模型左侧软件研发与右侧软件测试一一对应，每一层级测试用例对需求的覆盖率 T 反映了软件测试设计对软件显现需求的充分性。覆盖率越高，测试过程覆盖设计文档越全面，能确保软件是否完整实现了用户需求和设计。因此，测试用例对需求的覆盖率在一定程度上影响和反映了软件的质量。

开发各阶段对应的测试验证确保开发各个阶段的正确性，因此，常规指标中需求覆盖率指标能确保左侧软件开发各阶段的充分性，需求与测试用例的覆盖率可确保测试设计的充分性，软件代码覆盖率可确保测试执行的充分性。同时，软件开发过程从需求到代码，再到测试，形成环形追踪覆盖，因此，RTC 三项指标在 V 模型理论框架下可整体反馈软件测试质量。

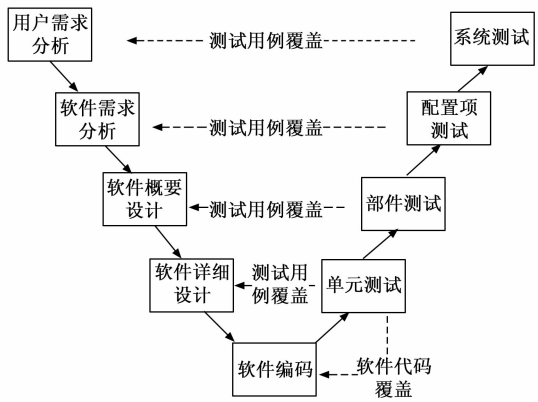

图 2　V 字开发模型

2.3　RTC 闭环覆盖率应用方向

RTC 覆盖率采集了软件需求、代码、测试三大环节追踪性指标，贯穿了软件研发过程。常用软件测试质量指标采用代码覆盖率，RTC 覆盖在其基础扩展到软件需求、测试设计，较为系统性反馈了软件测试质量。因此 RTC 可应用于自动化测试系统，在需求、测试设计、测试执行等环节设置采集指标，自动化测试结束后通过 RTC 闭环覆盖率体反应测试充分性。

3　基于 RTC 闭环覆盖率驱动的自动化测试系统设计

大多数自动化系统，仅实现了自动化测试脚本设计、测试执行功能，未设计测试需求管理、测试用例管理功能，以及代码覆盖率统计功能，导致缺乏软件需求与测试用例、测试用例执行与软件代码覆盖情况的统计，无法控制软件测试质量。针对以上问题，提出基于 RTC 闭环覆盖率驱动的自动化测试系统设计方法，通过设计软件需求管理、测试用例管理、被测软件版本管理、自动化测试执行等功能模块，实现软件工程化管理和软件开发持续集成，同时，增加软件需求、测试用例、代码等在需求颗粒度、用例密度、缺陷率、映射关系、RTC 覆盖率等方面的测量与统计。自动化测试系统采用分层和模块化设计，划分为管理与集成层、测试执行层、驱动层，如图 3 所示。

3.1　管理与集成层

管理与集成层具有软件需求管理，测试用例设计，自动化测试脚本设计，被测软件版本管理，持续集成能力，测试质量数据采集统计。软件需求管理包括：需求

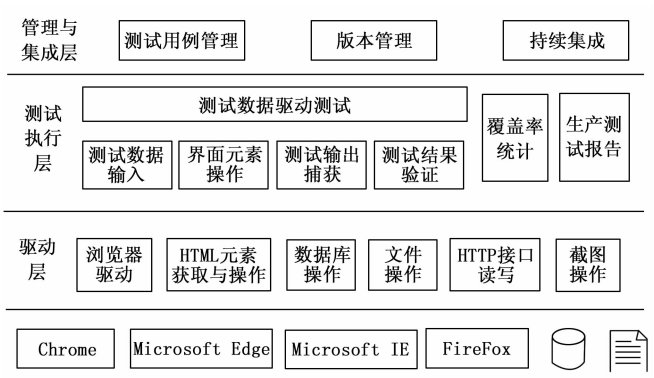

图 3 自动化测试系统分层级结构

名称，标识，需求描述，需求跟踪等功能；测试用例管理包括：测试用例描述，测试用例步骤，与软件需求追踪，测试自动化脚本映射，实测结果获取等功能；自动化测试脚本设计包括：测试脚本维护，编写，与测试用例映射等；版本管理包括：软件版本维护，代码检入，检出，导出，备份等；持续集成包括：任务自动执行策略编排，实现从版本管理模块获取被测代码，编译，部署，自动化测试脚本执行，测试结果和被测代码覆盖率统计输出等；测试质量数据采集统计包括：需求颗粒，用例密度，缺陷率，软件需求与代码覆盖率 R，软件需求与测试用例覆盖率 T，软件代码与测试用例覆盖率 C 等数据统计与测量。

3.2 测试执行层

测试执行层响应持续集成中的任务调用，执行测试用例所对应的自动化测试脚本，并调用驱动层实现测试数据输入、被测对象元素操作、测试结果获取，以及与预期结果的断言，最后将测试结果写入测试报告。同时，测试执行层具备测试流程驱动测试、测试数据驱动测试的能力。

此外，测试执行层通过代码覆盖率统计功能实现软件需求与代码对应功能，以及自动测试执行全部完成后的代码覆盖率统计功能。自动化测试脚本执行后，代码覆盖率统计模块将被执行的代码进行标注、入库，并与测试用例、软件需求对应，全部执行完成后将被测软件覆盖率情况、已执行语句、判断分支等情况写入覆盖率统计报告。

3.3 驱动层

驱动层响应测试执行层调用，主要为测试执行层提供操作各类浏览器的驱动，实现测试脚本对 HTML 元素定位与操作；提供数据库读写引擎，实现数据库查询、读写等；提供文件读写引擎实现 Txt、word、Excel、CVS 等文本读写功能。

4 原型验证系统的实现

基于 TestLink、Jenkins、TestNg、Selenium、SVN、Jacoco 等开源软件，通过二次开发、集成实现基于 RTC 闭环覆盖率驱动的自动化测试演示验证系统，实现需求管理、测试用例设计、测试脚本设计、代码管理、自动化测试集成、测试质量数据采集统计。

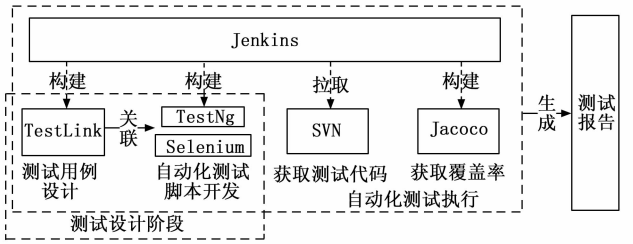

图 4 组件集成示意图

4.1 相关集成工具

4.1.1 Selenium

Selenium 是一个 Web 应用程序自动化测试框架，它通过模拟用户操作，如单击、下拉选择、文本输入、提交表单等操作，驱动浏览器进行功能自动化测试。测试人员使用 Selenium 编写自动化测试用例，通过重复运行测试用例提高测试效率。Selenium 支持多种浏览器，包括 Chrome、Firefox 和 Safari 等。

4.1.2 Jacoco

Jacoco 是一种 Java 代码覆盖率统计工具，在被测软件运行过程中统计被执行代码语句、分支，生成测试覆盖率报告，包括代码覆盖率、分支覆盖率和行覆盖率等。可帮助测试人员更好地了解其代码的测试覆盖情况，用于来评估软件测试的质量。

4.1.3 TestNg

TestNg 是一个基于 Java 的测试框架，可实现数据驱动测试、参数化测试、测试用例集测试、并发测试等功能，可以用于单元测试、集成测试和黑盒测试等。同时，TestNg 支持多线程执行测试用例，可以进行测试套件和测试分组，支持测试报告的生成和展示。

4.2 管理与集成层实现

管理与集成层通过安装 TestLink、SVN、Jenkins 工具，实现需求管理、测试用例设计、测试任务调度，以及被测代码管理版本管理等功能。TestLink 提供软件需求管理、测试用例设计功能，以及测试用例与自动化脚本的挂接功能，同时实现测试用例与需求的追溯统计，可转化为 RTC 模型的软件需求与测试用例覆盖率 T。通过二次开发将每个测试用例对应脚本的覆盖语句录入 TestLink 数据库，与软件需求形成追溯关系，即 RTC 模型的软件需求与代码的覆盖率 R。SVN 提供软件代码管理，具备软件代码检入、检出、导出等功能。Jenkins 可构建执行场景和步骤，实现从 SVN 中拉取被测软件，执行编译、部署，然后调用 TestNG 框架执行自动化测试脚本，将测试结果推送至 TestLink，最后调

用 Jacoco 输出覆盖率报告。

通过对 TestLink 扩展开发实现软件需求、测试用例、代码等在需求颗粒度、用例密度、缺陷率、映射关系、RTC 覆盖率等方面的统计，并形成相关统计报告。

4.3　测试执行层演示验证实现

测试执行层通过安装 TestNg 框架、Jacoco 工具，响应 Jenkins 任务调度，实现自动化测试脚本执行，代码覆盖率统计。TestNg 作为自动化测试脚本框架，具备数据驱动测试和过程驱动测试功能。Selenium IDE 录制测试过程，输出 Java 语言的自动化测试脚本，导入 TestNg 框架，挂接到对应的测试用例，供 Jenkins 任务构建调度。Jacoco 工具提供被测代码覆盖率收集功能，对部署在 Tomcat 的被测软件插入统计被执行代码，持续监视被测软件运行，在测试结束后响应输出命令导出被测代码覆盖率报告。

4.4　驱动层演示验证实现

驱动层主要通过 Selenium API 库、浏览器（Chrome、Firefox、IE）驱动、读取 mySql、Excel、CSV 文件的自定义开发类等构成，集成到 TestNg 框架中供执行层调用，以及管理与集成层调用。

4.5　基于 RTC 自动化测试系统操作流程

测试设计阶段：在 TestLink 中录入被测软件功能需求，然后测试人员根据录入的需求在 TestLink 软件编写测试计划、设计测试用例。测试人员依据测试用例设计，使用 Selenium IDE 浏览器插件，录制人工执行测试用例过程，导出 java 语言的自动化脚本，并设置输入数据和断言，并导入到 TestNg 测试框架中。最后在 TestLink 中将具体的测试用例与 TestNg 设计的对应测试脚本关联，形成从用户需求到测试用例，再到对应测试的关联。

测试执行阶段：在 Jenkins 软件中安装 TestLink、TestNg、Jacoco 插件，并配置构建过程，依次调用 SVN 拉取最新软件源代码，调用 TestNg 执行自动化测试脚本，过程中 Jacoco 被触发进行覆盖率收集，最后 Jenkins 将自动化测试脚本结果，即通过情况，写入 TestLink 中对应的测试用例执行结果，输出用例通过情况和软件代码覆盖率结果的测试报告。

测试人员根据 Jacoco 覆盖率统计情况，分析未覆盖原因，增加测试用例，以提高覆盖率的方式，防止漏测。

5　实验设计和结果分析

5.1　试验目的

本章节以验证软件数字大小比较功能为例，使用基于 RTC 驱动的自动化测试原型系统，微型化推演需求管理、用例设计、自动化脚本编写、测试执行、生成测试报告等过程，展示试验结果。

5.2　试验内容

在基于 RTC 驱动的自动化测试原型系统，试验“比较两个输入的数据，返回最大值”软件功能需求分析、编码、用例设计、自动化测试全生命周期过程，以及质量数据量化采集。

5.3　试验步骤

5.3.1　需求分析

在 TestLink 软件需求管理模块，新建“比较两个输入的数据，返回最大值”的概要需求，并分解出两个子功能需求：

1）不相同的两个整数返回最大值；

2）相同的两个整数返回任意一个整数。将以上分解的子功能需求的标识、名称、需求描述录入需求管理模块。

5.3.2　软件实现编码

软件代码，如下：

```
public class cc {
public int reMaxValue(int a,int b)
{
   if(a>b)
     return a;
   else if(a==b)
     return a;
   else
     return b;
}
}
```

5.3.3　测试用例设计

在 TestLink 软件测试用例管理模块，针对子功能需求设计测试用例，并录入用例名称、用例说明、测试步骤、测试执行类型（自动）、关联具体的子需求、关联用例执行的脚本等。

1）子功能需求：不相同的两个整数返回最大值。

测试用例 01：输入测试值 1，2，预期返回 2；

测试用例 02：输入测试值 5，4，预期返回 4；

2）子功能需求：相同的两个整数返回任意一个整数。

测试用例 03：输入测试值 6，6，预期返回 6；

在 TestLink 软件完成用例设计后，可调用需求管理模块的概览，查看各个需求点是否用测试用例覆盖，以达到用户需求 100％覆盖。

5.3.4　开发自动化测试脚本

通过 TestNg 框架集成 Selenium 库、Chrome API 实现操作浏览器，在“第一个数字”，“第二个数字”输入框输入值，并点击“判断最大数”按钮，获取“最大数为：”输出标签的值，与预期结果比较。如果实测值

与预期一致，则用例通过，否则用例失败。

以“测试用例 01：输入测试值 1，2，预期返回 2”为例，测试脚本如下：

```
public class Test_01 {
    private WebDriver driver;
    private Map<String, Object> vars;
    JavascriptExecutor js;
    @BeforeMethod
    public void setUp() {
    System.setProperty("webdriver.chrome.driver", "D:\chromedriver.exe");
    driver = new ChromeDriver();
      js = (JavascriptExecutor) driver;
      vars = new HashMap<String, Object>();
  }
    @AfterMethod
    public void tearDown() {
      driver.quit();
  }
    @Test
    public void test() {
      driver.get("http://localhost:8080/maxValue_war/index.html");
       driver.manage().window().setSize(new Dimension(1157, 774));
       driver.findElement(By.name("num1")).sendKeys("1");
       driver.findElement(By.name("num2")).sendKeys("2");
      driver.findElement(By.cssSelector("p:nth-child(3) > input")).click();
      assertThat(driver.findElement(By.id("returnValue")).getText(), is("2"));
    }
}
```

5.3.5 自动化执行配置

1）配置 Maven：

通过 Testng 框架中 Testng. xml 将测试用例集中管理起来，并利用 Maven 实现 Testng 测试框架、自动化测试脚本的编译，以及按照 Testng. xml 中的用例集执行。在 Testng. xml 配置执测试用例 01、测试用例 02。

2）测试用例与脚本映射：

在 TestLink 软件中，通过自定义字段，映射测试用例与自动化测试脚本。比如：在 TestLink 软件中自定义字段为 result，将“测试用例 01”自定义字段 result 设置为 Test_01#test。后续 Jenkins 调用 TestLink 插件将自动化自行结果反向写入 TestLink 的“测试用例 01”用例的执行结果，以此配置“测试用例 02”。

3）配置 Jacoco 覆盖率监视：

被测 web 软件通过 Tomcat 提供访问服务。在 Tomcat 的启动批处理“catalina.bat”中配置 Jacoco agent 启动命令，实现 Jacoco agent 对被测 web 软件的监视，并提供 TCP 端口响应请求输出覆盖率记录文件的命令。

4）配置 Jenkins：

在 Jenkins 中，新建 Maven 工程，并在工程各个环节配置参数，实现自动化代码的获取、编译、部署、Jacoco 监视启动、自动化测试程序运行、Jacoco 覆盖率文件导出和发布、自动化脚本测试结果反向关联 TestLink 测试用例。具体配置如下：在“General”设置 TestNg 自动化框架项目工作路径；在“源码管理”设置 GIT 的获取路径、许可等；“Pre build”中配置被测源码的编译命令，将生成的 war 包，拷贝值 Tomcat 的 webapps 部署；“build”中配置启动自动化测试脚本编译、运行的 POM 路径；“Post Steps”配置 window 环境批量执行命令，从 Jacoco agent 端口导出被测 web 软件在自动化测试后的覆盖率文件；“构建后操作”配置分析、发布覆盖率命令；再增加一组“构建后操作”，通过 TestLink 插件将 TestNg 各自动化脚本执行结果挂接到 TestLink 软件对应的测试用例执行结果中。

5）自动化测试执行：

在 jenkins 中，设置定时运行配置项的任务，启动测试脚本的自动化执行。

5.4 测试结果及分析

5.4.1 测试用例自动执行结果

测试结果可在 TestLink 软件 Report 模块中查看各用例实测值、通过情况，本次用例全部通过，以及关联的需求等等，如图 5 所示。

Test Case Pro01-1: 测试用例_1 [Version : 1]			
Summary:	测试输入1,2，预期返回2。		
#:	Step actions:	Expected Results:	Execution Status:
1	输入测试值，1,2。	预期返回2.	
Execution type:	Automated		
Estimated exec. duration (min):	2.00		
Priority:	Medium		
Execution Details			
Build	1		
Tester	admin		
Execution Result:	**Passed**		

图 5 测试用例执行结果

5.4.2 覆盖率结果查看

测试覆盖率可查看 Jacoco 生成的覆盖率结果网页，

浏览总体被测代码覆盖率情况，以及未覆盖的具体代码。本次执行 2 个测试用例，代码覆盖率 66.67%。

图 6　代码覆盖情况

5.4.3　测试结果分析

调用统计报表展示了工程软件需求个数、代码行数、执行用例数，以及测量的需求密度 333.3 个/行，*RTC* 值为 66.67%，表明软件测试质量存在潜在问题，其中 *C* 值未达到 100%表明代码未覆盖完全。

软件需求个数：	2个	R值：	100%
代码行数：	6行	T值：	100%
测试用例：	3个	C值：	66.67%
执行测试用例：	2个	RTC值：	66.67%
需求颗粒度：	333.3个/kloc		

图 7　工程度量统计

5.5　后续提升思考

在本次实验中，通过搭建的集成 Jenkins、TestLink、TestNg、Jacoco、Selenium 等工具，结合二次开发实现测试需求管理、测试用例管理、测试脚本开发、自动化测试执行、覆盖率实时统计等功能，并通过覆盖率统计，二次开发实现软件需求与功能的映射，以及需求颗粒，用例密度，缺陷率，软件需求与代码覆盖率 *R*，软件需求与测试用例覆盖率 *T*，软件代码与测试用例覆盖率 *C* 等数据统计与测量。并通过 *RTC* 闭环覆盖率指标，反馈测试质量，驱动自动化测试改进，从而提高自动化测试质量和效率。但系统未实现灵活、配置测试用例执行集合，并产生对应执行结果和覆盖率情况。在此研究基础上，后续可探索通过逐个执行测试，统计出测试需求对应的具体软件代码，用于在软件代码变更后分析影响的测试用例和测试需求，指导回归测试用例选取、修改或增加。

6　结束语

该自动测试系统贯通了软件开发的需求设计、测试设计、测试执行等环节，自动化测试执行提高了测试效率，同时也提供了监视测试质量的指标，但过程中发现了本框架一些不足和局限。首先，软件需求功能对应的代码映射，包含了很多重复代码；其次，本框架只考虑了代码行数覆盖率和分支覆盖率，未来可以考虑更多的覆盖率指标；再次，自动化测试系统集成测试用例、测试脚本设计较松散，不变使用，应该通过深度二次开发将需求设计、测试设计、自动化脚本设计、任务管理集成到一个软件。

综上所述，提出的基于 RTC 闭环覆盖率驱动的自动化测试系统在实际应用中具有广泛的应用前景。希望未来能有更多的研究者和开发人员加入进来，共同推动自动化测试技术的发展和应用。

参考文献：

[1] 姜　文，刘立康. 基于 Selenium 的 Web 软件自动化测试[J]. 计算机技术与发展，2018，28 (9)：47－52.

[2] 朱少民. 软件测试方法和技术 [M]. 北京：清华大学出版社，2014.

[3] 李雨江. 代码覆盖率工具 BullseyeCoverage 的应用研究[J]. 湛江师范学院学报，2013，34 (6)：101－104.

[4] 梁　俊. 基于 Selenium 和 TestNG 的自动化测试框架的设计与实现 [D]. 青岛：山东科技大学，2018.

[5] 吴应福. 软件自动化测试方法及其发展现状实践分析[J]. 市场周刊理论版，2017 (28)：256－256.

[6] 吴　军. 智能家电控制器自动化测试系统设计与实现分析[J]. 仪器仪表标准化与计量，2020 (6)：42－43.

[7] 龚智勇. 基于 Selenium 的 OpenStack Horizon 自动化测试的实现 [J]. 国外电子测量技术，2017，36 (5)：45－49.

[8] 樊庆林，吴建国 . 提高软件测试效率的方法研究 [J]. 计算机技 术与发展，2006，16 (10)：52－54.

[9] 张　瑾，杜春晖 . 自动化软件测试 [M]. 北京：机械工业出版社，2004.

[10] 黄华林. 使用 Selenium 进行 Web 应用自动化测试的研究[J]. 电脑开发与应用，2012，25 (4)：54－56.

[11] 安金霞，王国庆，李树芳，等. 基于多维度覆盖率的软件测试动态评价方法 [J]. Journal of Software，2010，21 (9)：2135－2147.

[12] PERRY W E. Effective methods for software testing [M]. 3rd ed. Beijing：Tsinghua University Press，2008 (in Chinese).

[13] 唐　军. 测控站的自动化测试及标校 [J]. 电讯技术，1995，35 (6)：12－18.

[14] 张元华，王　峻. 通过软件测试提高航空电台软件质量[J]. 电讯技术，2006，46 (3)：170－172.

[15] FRIEDENTHAL S，MOORE A，STEINER R. Integrating SysML into a systems development environment. A Practical Guide to SysML. 2nd ed. [S. l.]：Elsevier，2012：523－556.

（下转第 44 页）