文章编号: 1001-4543(2022)02-0164-08 DOI: 10.19570/j.cnki.jsspu.2022.02.011

基于持续测试技术的整车控制器测试平台研究

陈振宇[1], 秦 琴[1], 丁 锋[2], 吴 辉[2]

(1. 上海第二工业大学 智能制造与控制工程学院, 上海 201209; 2. 联合汽车电子有限公司, 上海 201206)

摘 要: 针对当前整车控制器软件测试效率低下和测试任务执行不及时问题, 设计了一个基于持续测试技术的整车控制器测试平台。平台自动化完成待测软件获取、测试脚本筛选、测试环境部署、测试任务执行以及测试报告发送等整车控制器测试流程, 同时根据具体需求设计了多测试任务调度算法实现测试任务优化分配。实验结果表明, 持续测试平台测试效率相较于现有自动测试平台测试效率提高了 14%, 测试稳定性提高了 7%, 持续测试平台整体性能优于现有自动测试平台。

关键词: 持续测试; 测试效率; 多测试任务调度算法; 新能源汽车; 整车控制器

中图分类号: U463.6 **文献标志码**: A

Research on Vehicle Controller Test Platform Based on Continuous Test

CHEN Zhenyu[1], QIN Qin[1], DING Feng[2], WU Hui[2]

(1. School of Intelligent Manufacturing and Control Engineering, Shanghai Polytechnic University, Shanghai 201209, China; 2. United Automotive Electronic Systems Co., Ltd., Shanghai 201206, China)

Abstract: Aiming at the current problems of low efficiency and untimely execution of vehicle controller software testing, a vehicle controller testing platform based on continuous testing technology is designed. The vehicle controller test processes such as software acquisition, test script screening, test environment deployment, test task execution and test report sending were completed through the technology of platform automation, and the multi test task adjustment algorithm was used to optimize the assignment of test tasks according to the specific requirements. The experimental results show that the test efficiency of continuous test platform is 14% higher than that of existing automatic test platform, the test stability is 7% higher, and the overall performance of continuous test platform is better than that of existing automatic test platform.

Keywords: continuous testing; test efficiency; multi test task adjustment algorithm; new energy vehicle; vehicle controller

0 引言

随着敏捷和 DevOps 开发模式在软件行业的推广落地, 更频繁的交付加重了业界对测试的担忧, 测试不够高效往往成为导致交付延期的首要原因, 测试环节也成为了企业进行 DevOps 转型的最大瓶颈。为了应对这种挑战, “持续测试” 概念逐渐被汽车行业广泛接受 [1]。目前, 新能源汽车整车控制器的测试领域主要存在以下两大难题。第 1 个难题是使用自动测试平台测试整车控制器, 测试过程仍需测试工程师手工参与。测试工程师需要提前对测试环境进行配置, 然后从测试脚本管理库中筛选测试脚本, 并将其放置在测试环境中, 最后触发自动测试 [2-3]。不同的测试内容需要测试工程师不断地筛选测试脚本和配置环境, 这就导致了测试效率低下, 消耗了太多的测试工程师的精力, 使测试过程容易

收稿日期: 2021-09-25

通信作者: 秦 琴 (1978–), 女, 辽宁沈阳人, 副教授, 博士, 主要研究方向为智能检测与运动控制。E-mail: qinqin@sspu.edu.cn

基金项目: 电子信息类专业硕士协同创新平台建设 (A10GY21F015) 资助

出现疏漏。另一个难题是整车控制器软件测试不及时。软件开发工程师希望在完成软件开发之后尽快开始测试操作, 并且希望尽早发现整车控制器软件中的错误。然而, 测试工程师的精力有限, 需要测试工程师完成现有的测试任务后才能执行新触发的测试任务。测试任务触发与实际执行测试任务之间的时间受到测试工程师任务量的影响, 测试任务无法及时执行。

为了保证软件开发工程师们触发的测试任务能够及时被执行, 测试结果能够及时反馈给相应的软件开发工程师, 同时也为了进一步提高测试设备使用率和整车控制器软件测试效率, 保证软件质量和缩短软件开发周期。本文设计了一个基于持续测试技术的整车控制器测试平台, 在完成整车控制器软件的编译后, 自动触发整车控制器的软件测试过程, 如果测试任务的数量大于测试设备的数量, 则对测试设备进行调度管理, 使软件工程师能够尽快了解整车控制器软件中的错误信息, 修补软件中的漏洞。该持续测试平台对于提高设备使用率和测试效率, 解放人力, 缩短软件开发周期, 提高整车控制器软件质量具有重要意义, 并为其他控制器的持续测试发展提供参考与理论基础。

1　持续测试技术概述及原理

持续测试技术是一种以并行的方式对开发中的软件进行实时测试的过程, 持续测试技术使软件开发与软件测试活动保持一致, 以此向软件开发工程师提供软件错误信息的一种技术理论, 持续测试技术的目的是在软件开发的早期发现软件中的问题, 以此避免软件中出现不易察觉的错误出现 [4]。

在整车控制器软件开发与测试中使用持续测试技术时, 需要软件工程师根据开发需求进行任务分解, 将测试任务分解成一个个具体的开发任务。当软件开发工程师开发完第 1 个子任务后, 测试工程师对其进行测试, 软件开发工程师在等待测试结果的时候, 可以继续进行第 2 个子任务开发。第 1 个子任务测试完成后, 测试工程师将软件中的问题反馈给软件开发工程师进行修复, 直到软件测试通过。随后, 软件开发工程师的第 2 个子任务开发完成, 测试工程师能够立即进行第 2 个子任务的软件测试, 软件开发工程师则进行第 3 个子任务的开发。以此类推, 开发和测试同时进行, 所有项目人员在软件开发的每个中间环节能参与其中, 整车控制器持续测试技术原理如图 1 所示。

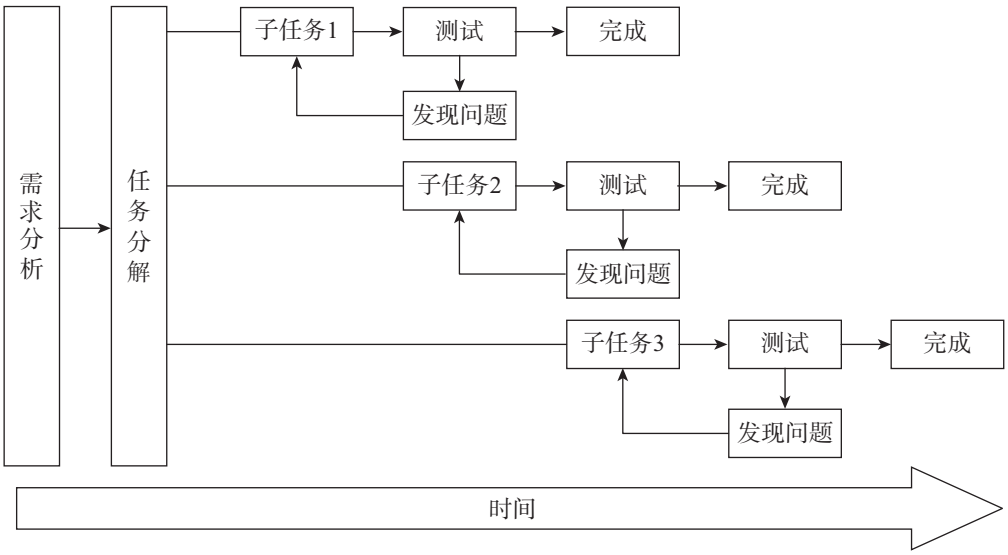

图 1　整车控制器持续测试技术原理

Fig. 1　Technical principle of continuous testing of vehicle control unit

2　持续测试平台构成

持续测试平台分为硬件和软件两部分, 硬件部分包括服务器主机、客户端主机、测试设备和整车控制器; 软件部分包括 RQM 任务管理系统、测试脚本管理库和 Jenkins 主从程序。服务器主机上运行 Jenkins 主程序 (Jenkins-Master), 服务器主机作为平台的任务调度中心, 监控待测软件的提交, 触发整车控制的测试和管理测试结果, 保证持续测试顺利运行。测试脚本管理库存放从 RQM 任务管理系

统生成的测试脚本。客户端主机上运行 Jenkins 从程序 (Jenkins-Slave), 接收 Jenkins-Master 发送过来的控制指令。当有待测软件被 Jenkins-Master 检测到时, Jenkins-Slave 根据 Jenkins-Master 的指令开启 ECU-TEST, 触发测试设备进行整车控制器软件测试 [5]。测试完成后 ECU-TEST 生成测试报告, 并通过 Jenkins-Master 上传到 RQM 中进行测试结果管理。持续测试平台的系统结构如图 2 所示。

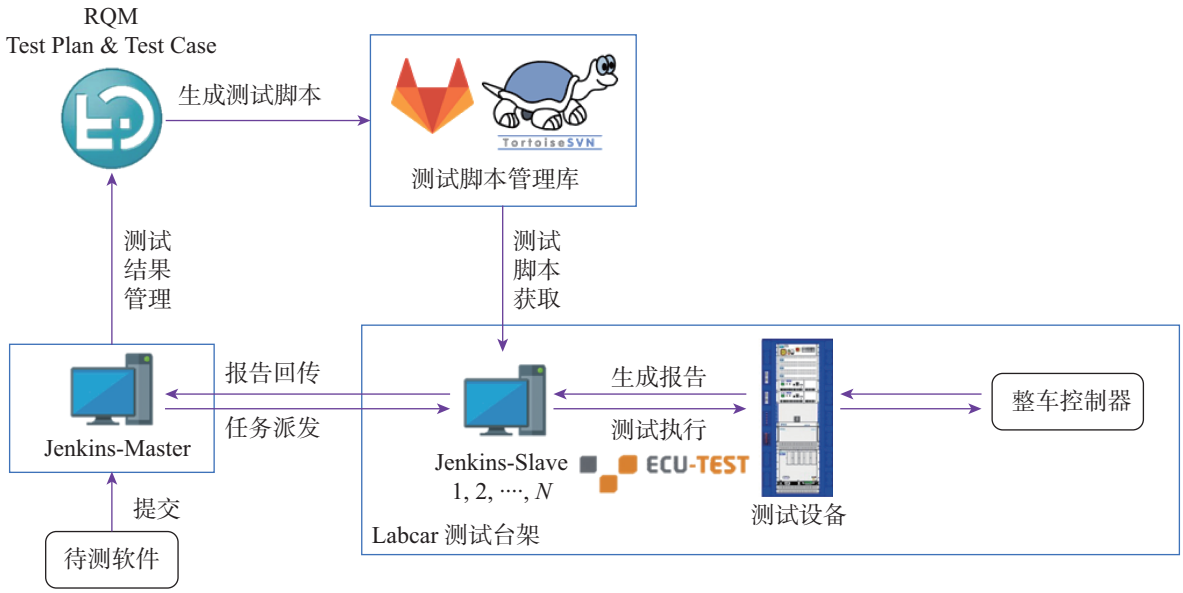

图 2 持续测试平台系统结构图

Fig. 2 System structure of continuous testing platform

3 持续测试平台基本测试流程

持续测试平台是对自动测试流程的优化, 因此在进行持续测试平台测试流程设计前, 需要了解自动测试基本流程。自动测试的硬件设备 Labcar 测试台架 (见图 2), 在进行整车控制器自动测试时, 软件开发工程师通过邮件将需要的软件发送给测试工程师。测试工程师在获得软件后, 根据软件需要测试的内容在测试脚本管理库中筛选出使用的测试脚本。测试工程师将待测软件和测试脚本放到 ECU-TEST 的配置文件中, 随后 ECU-TEST 对测试环境进行部署。测试环境部署完成后, 测试工程师通过在 ECU-TEST 上手动运行相对应的测试脚本, 触发自动测试流程 [6]。测试完成后, 测试工程师对测试报告进行整理, 将测试报告反馈给软件开发工程师, 自动测试基本流程如图 3 所示。

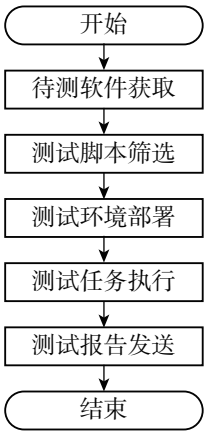

图 3 自动测试基本流程

Fig. 3 Basic process of automatic test

持续测试平台进行测试的流程如图 4 所示。在进行测试任务之前, 测试工程师需要对 RQM 上登记的测试计划和测试任务进行检查和维护, 并将测试计划和测试脚本放置在 GIT 或者 SVN 中; 服务器上软件编译完成后被 Jenkins-Master 识别, Jenkins-Master 根据生成的待测软件信息在 RQM 上查询相应的测试计划和测试任务; Jenkins-Master 根据测试任务和测试计划, 指派当前可以使用的 Jenkins-Slave 从 GIT 或者 SVN 上下载相应的测试脚本到本地; 测试脚本下载完成后, Jenkins-Slave 控制客户端电脑对 ECU-TEST 和 Labcar 进行测试环境的部署; 测试环境部署完成后, ECU-TEST 按照测试脚本内的测试流程对整车控制器进行相应的刷新和测试操作。Labcar 测试完成后, ECU-TEST 生成测试数据和测试报告; 生成的测试数据和测试报告被储存在 Jenkins-Master 服务器中, 同时测试数据

被存入到网盘中; Jenkins-Master 将网盘地址上传到 RQM 中进行管理, 同时将测试报告的网盘路径发送给相应工程师, 由工程师对测试结果进行检查; 最后, 持续测试平台等待下一个测试任务触发。

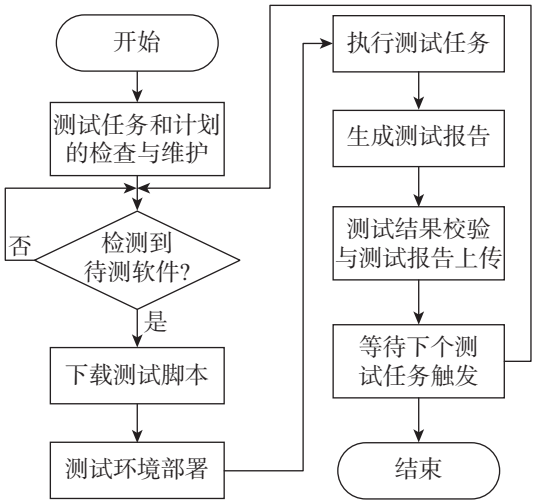

图 4　持续测试平台的基本测试流程图

Fig. 4　Test flow chart of continuous testing platform

4　多测试任务调度算法

在持续测试平台执行整车控制器测试的任务中, 一个测试任务可以按照具体测试内容拆分成多个子测试任务, 子测试任务之间测试流程相互独立, 因此测试任务可以拆分成多个子测试任务, 安排多台 Labcar 测试台架对其进行测试。软件开发的生命周期越来越短, 测试任务越高效, 整车控制软件质量才能得到保障。因此, 需要有一套高效的实时测试调度系统来提高测试效率, 减少测试结果等待时间。在计算机操作系统中经常会遇到资源调度问题, 并且也有很多成熟稳定的解决算法, 银行家算法是其中经典算法之一 [7]。本文使用银行家算法来解决持续测试平台的测试设备调度问题 [8-9]:

设 Req_i 为进程 T_i 的请求向量, 如果 $\mathrm{Req}_i[j] = K$, 表示进程 T_i 需要 K 个 R_j 类型的资源。当 T_i 发出资源请求后, 系统按照下列步骤进行检查:

(1) 如果 $\mathrm{Req}_i[j] \leqslant$ 需求向量 $\mathrm{Need}[i,j]$, 便转向步骤 (2), 否则认为出错, 因为它所需要的资源已经超过它所宣称的最大资源。

(2) 如果 $\mathrm{Req}_i[j] \leqslant$ 可利用的测试设备数量 $\mathrm{Available}[j]$ 便转向步骤 (3), 否则表示尚无足够的 Labcar 资源, T_i 须等待。

(3) 系统试探着把资源分配给进程 T_i, 并修改下面结构中数据的数值:

$$\mathrm{Available}[j] = \mathrm{Available}[j] - \mathrm{Req}_i[j] \tag{1}$$

$$\mathrm{Allocation}[i,j] = \mathrm{Allocation}[i,j] + \mathrm{Req}_i[j] \tag{2}$$

$$\mathrm{Need}[i,j] = \mathrm{Need}[i,j] - \mathrm{Req}_i[j] \tag{3}$$

(4) 如果 $\mathrm{Need}[i,j] \leqslant \mathrm{Available}[j]$, 系统认为将 Labcar 测试资源分配给该测试任务可行, 测试任务 T_i 获得此次分配的 Labcar 的使用权限, 否则将本次的试探分配作废, 恢复原来的 Labcar 分配状态, 让测试任务 T_i 等待。

(5) 当按照银行家算法分配完 Labcar 测试设备后, 如果还有可以使用的 Labcar 测试设备, 则将剩余的 Labcar 测试设备平均分配给剩下的测试任务使用。

基于银行家算法, 对测试任务进行优先级排序, 在环仿真系统为测试任务分配最佳的硬件。假设现有 5 个硬件在环仿真系统, 某个时间段有 3 个整车控制器软件测试任务需要进行系统测试, 其测试队列的状态如表 1 所示。其中第 1 个测试任务 T_1 包含整车控制器引脚的输入和输出测试等 2 个系统测试子任务, 第 2 个测试任务 T_2 包含整车控制器引脚的输入、输出测试、通信发动和通信接收等 4 个系统测试子任务, 第 3 个测试任务包含整车控制器通信发动和通信接收等 2 个系统测试子任务 (见表 2)。

表 1　某时刻持续测试平台测试队列初始状态

Tab. 1　Initial state of test queue of continuous test platform at a certain time

测试任务	资源情况				
	测试任务数量	分配数量	需求数量	完成测试数量	测试状态
T_1	2	0	2	0	测试未完成
T_2	4	0	4	0	测试未完成
T_3	2	0	2	0	测试未完成

表 2 持续测试平台分配方案 1

Tab. 2 Continuous testing platform allocation scheme 1

测试任务	资源情况				
	测试任务数量	分配数量	需求数量	完成测试数量	测试状态
T_1	2	2	0	0	测试未完成
T_2	2	2	0	0	测试未完成
T_3	4	1	3	0	测试未完成

为了保证持续测试平台能测试尽可能多的测试任务, 测试平台根据银行家算法对测试任务进行优先级排序。在测试任务列表中 T_1 包含两个测试任务, T_2 包含 4 个测试任务, T_3 包含 2 个测试任务。此时测试资源分配 (见表 2)。

当 T_1 和 T_3 的测试任务完成后, 持续测试平台将剩下所有可用的 Labcar 分配给 T_2 进行测试。此时测试资源分配如表 3 所示。

表 3 持续测试平台分配方案 2

Tab. 3 Continuous test platform allocation scheme 2

测试任务	资源情况				
	测试任务数量	分配数量	需求数量	完成测试数量	测试状态
T_1	2	0	0	2	测试完成
T_2	2	0	0	2	测试完成
T_3	4	4	0	0	测试未完成

最终持续测试平台完成 T_1、T_2 和 T_3 测试任务, 相较于没有 Labcar 资源调度系统前, 当某个时间段有大量测试任务需要进行测试, 测试任务被拆分成多个子任务进行并行测试, 单个测试任务的测试效率得到了提升。

5 持续测试平台使用

在整车控制器软件测试中, 测试内容主要包含软件刷新测试、IO 接口测试和通信测试 3 个主要步骤, 本文的持续测试平台的使用过程以 IO 接口测试为例进行介绍。持续测试流程在前文已经详细叙述, 此处不再进行赘述。

使用持续测试平台时, 测试工程师只需要提前在 RQM 中登记测试的项目名称、测试脚本路径、模型文件路径和相关工程师的邮箱, 然后等待服务器中生成的测试软件被 Jenkins-Master 检测到。RQM 登记测试任务信息界面如图 5 所示。

图 5 RQM 登记测试任务信息界面

Fig. 5 RQM registration test task information interface

当 Jenkins-Master 检测到软件编译服务器的整车控制器软件后, Jenkins-Master 根据图 5 所示的登记信息, 选择对应的 Jenkins-Slave 开始进行整车控制器的测试流程 [10]。图 6 所示为持续测试触发后, Jenkins-Slave 控制 ECU-TEST 进行 IO 接口测试的过程。测试完成后, ECU-TEST 自动生成整车控制器 IO 接口的测试报告, 软件工程师可以点击具体的控制器引脚查看详细的测试结果。测试报告部分截图如图 7 所示。

持续测试平台在整个测试流程结束后, 将测试报告的路径上传至 RQM 任务登记界面, 同时 Jenkins-Master 通过邮件将测试报告路径发送给相关工程师查看测试报告。测试报告路径上传 RQM 如图 8 所示。

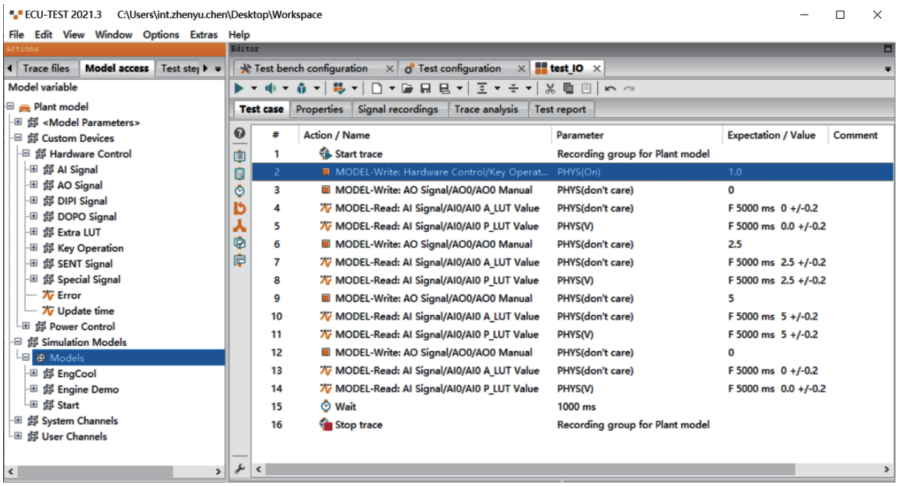

图 6　ECU-TEST 进行 IO 接口测试

Fig. 6　ECU-TEST performs IO interface test

18 (I_A_　　)

Model_stim:
Ecu_meas:
stimUnit:
measUnit:

User comments

No comments available

Stimulation	Expectation	Measurement	rel. Tolerance	abs. Tolerance	Exp. min.	Exp. max
0.00	0.00	0.00	2.00	0.05	-0.05	0.05
0.02	0.02	21.00	2.00	0.05	-0.03	0.07
2.50	2.50	2 503.00	2.00	0.05	2.45	2.55
4.98	4.98	4 983.00	2.00	0.05	4.88	5.08
5.00	5.00	4 999.00	2.00	0.05	4.90	5.10

图 7　测试报告部分截图

Fig. 7　Screenshot of test report

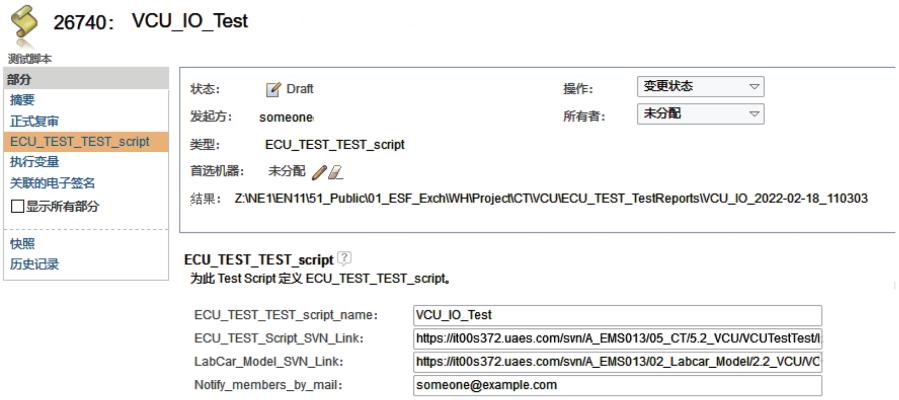

图 8　测试报告路径上传 RQM

Fig. 8　Test report path uploaded to RQM

6 测试结果与分析

从测试效率以及测试成功率两个维度对自动测试平台和持续测试平台进行比较。

(1) 测试效率对比。通过对整车控制器软件刷新测试、IO 接口测试和通信测试 3 种测试任务在相同的整车控制器、相同的测试脚本和相同的测试环境下进行了 20 次测试, 分别记录每个测试任务的在持续测试平台和自动测试平台上消耗的时间, 其结果如表 4 和图 9 所示。

通过相同测试脚本和测试环境, 得出持续测试平台对自动测试平台测试效率提升的程度, 得到如表 5 所示持续测试与自动测试相对效率表。即

$$\eta_1 = (t_z - t_c)/t_z \tag{4}$$

式中: η_1 为持续测试相对自动测试提高的测试效率;

表 4 持续测试平台和自动测试平台测试时长

Tab. 4 Test duration of continuous test platform and automatic test platform

测试序列	持续测试/s				自动测试/s			
	软件刷新测试	IO 接口测试	通信测试	测试总时长	软件刷新测试	IO 接口测试	通信测试	测试总时长
1	186	554	8 014	8 755	236	772	9 691	10 699
2	159	468	8 123	8 749	247	789	9 324	10 360
3	181	493	8 188	8 863	252	765	9 563	10 580
4	164	457	8 025	8 645	272	759	8 572	9 604
5	176	557	8 222	8 955	244	720	8 537	9 501
6	168	500	8 215	8 883	272	792	8 717	9 781
7	181	511	8 131	8 823	275	777	8 970	10 022
8	182	510	8 219	8 911	243	824	9 039	10 106
9	169	499	8 045	8 713	250	685	9 542	10 477
10	170	522	8 145	8 837	240	797	9 824	10 861
11	181	449	8 141	8 771	239	768	9 473	10 481
12	192	482	8 065	8 739	240	825	9 621	10 686
13	161	469	8 139	8 769	271	818	8 989	10 078
14	159	543	7 968	8 670	244	804	9 293	10 340
15	179	493	8 110	8 782	269	729	8 600	9 598
16	165	478	8 198	8 841	262	762	9 677	10 701
17	146	499	8 093	8 739	283	767	9 408	10 457
18	159	485	8 197	8 841	239	747	9 036	10 022
19	162	484	8 105	8 751	231	785	8 805	9 822
20	171	494	8 008	8 673	230	851	9 470	10 551

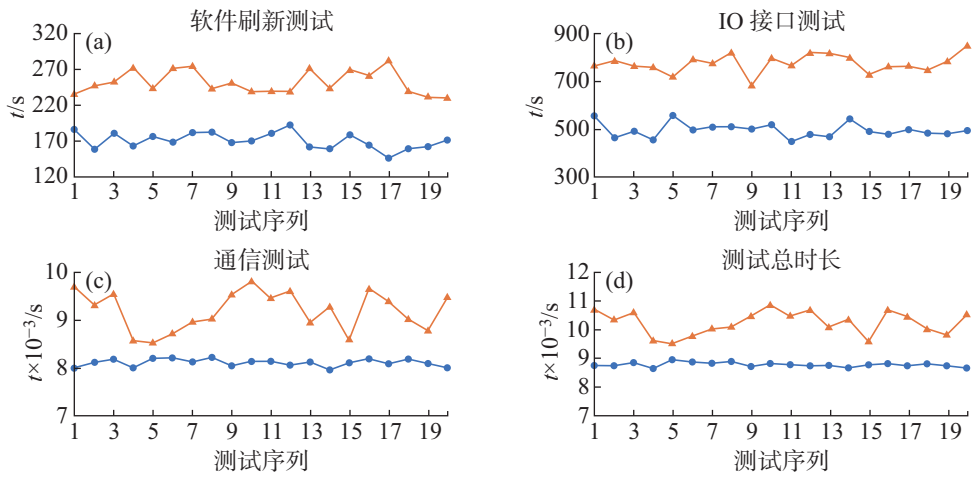

图 9 测试时长 (t) 对比

Fig. 9 Test time (t) comparison

t_z 为相同测试条件下自动测试所用时间, s; t_c 为相同测试条件下持续测试所用时间, s。

本文在相同测试脚本和测试环境下, 对相同整车控制进行了测试。测试结果表明, 持续测试平台所需时间比自动测试平台更少。持续测试平台相较于自动测试平台, 软件刷新测试、IO 接口测试和通信测试等任务的测试平均效率分别提高了 32.14%、36.03% 和 11.84%, 总体测试效率提高了约 14%。

表 5　持续测试与自动测试相对效率

Tab. 5　Relative efficiency of continuous test and automatic test

	测试类型			
	软件刷新测试	IO 接口测试	通信测试	测试总时长
持续测试时长/s	171	497	8 118	8 786
自动测试时长/s	252	777	9 208	10 236
测试平均效率提升/%	32.14	36.03	11.84	14.17

(2) 测试稳定性对比。通过对软件刷新、IO 接口和通信 3 种测试任务在相同整车控制器、相同测试脚本和相同测试环境下进行 100 次测试统计, 根据持续测试平台和自动测试平台执行测试的成功的次数比较两种测试平台的稳定性。计算得到持续测试平台与自动测试平台测试成功率比较 (见表 6)。实验表明, 持续测试平台比自动测试平台的稳定性要高出 7%。其公式表示为

$$\eta_2 = P/T \tag{5}$$

式中: η_2 为测试平台的稳定性; P 为测试成功的次数; T 为测试的总次数。

表 6　持续测试平台与自动测试平台测试成功率比较

Tab. 6　Comparison of test success rate between continuous test platform and automatic test platform

	测试平台	
	自动测试平台	持续测试平台
成功次数	91	98
失败次数	9	2
成功率/%	91	98

7　结　语

本文设计了一种基于持续测试技术的整车控制器测试平台, 该测试平台检测到待测整车控制器软件后, 自动开始测试流程, 测试完成后自动将测试报告上传到任务管理系统 RQM 中, 并将测试报告自动发送给整车控制器相关工程师。相较于传统自动测试平台, 本文设计的持续测试平台使用持续测试技术触发整车控制器测试流程, 利用多测试任务调度算法将测试任务分配到不同的 Labcar 测试台架中执行测试。实验证明, 整车控制器的测试效率提高了 14%, 测试稳定性提高了 7%。持续测试平台能充分利用 Labcar 测试资源, 控制器软件中的问题能够被及时发现, 软件开发周期得到了缩短, 整车控制器软件质量也得到了提高, 同时也为其他控制器的持续测试提供参考与理论基础。

参考文献:

[1] CODING_devops. 持续测试 DevOps 时代的高效测试之钥 [EB/OL]. [2021-05-31]. https://blog.csdn.net/ CODING_devops/article/details/117418584.

[2] 宋茜, 吴速超, 王龙, 等. 使用 ECU-TEST 进行 IO 接口批量自动化测试的研究 [J]. 汽车电器, 2020(11): 55-56.

[3] 徐永新, 朱娟, 王裕鹏. 基于 ECU_TEST 的 ECU 报文自动测试研究与应用 [J]. 汽车电器, 2018(10): 71-74.

[4] 张刚. 基于 Eclipse 平台的持续测试开发工具研究与实现 [D]. 长沙: 国防科学技术大学, 2004.

[5] 宋茜, 吴速超, 耿宗起, 等. 基于 ECU-TEST 和 Jenkins 的自动化测试方法研究 [J]. 汽车电器, 2021(5): 69-70.

[6] 马传福. 基于硬件在环系统的发动机 ECU 下线测试自动测试开发 [D]. 长沙: 湖南大学, 2017.

[7] 常立丹. 平面移动式立体车库 RGV 运行阻塞与死锁分析及对策研究 [D]. 兰州: 兰州交通大学, 2021.

[8] 宋丹, 李亚东. 银行家算法的改进与实现 [J]. 计算机时代, 2021(6): 64-67.

[9] 江立. 基于银行家算法的进程安全序列改进算法 [J]. 计算机产品与流通, 2019(5): 155-156.

[10] 程宁, 戴远泉. 基于 Jenkins 持续集成部署研究与实现 [J]. 电子制作, 2021(22): 46-48.