文章编号:1671-4598(2019)06-0055-07 DOI:10.16526/j.cnki.11—4762/tp.2019.06.013 中图分类号:TP311 文献标识码:A

航空机载软件测试工具链设计与应用

李 昌,蒋友毅,宋雁翔,冯 雷
(成都飞机设计研究所,成都 610091)

摘要:随着航空机载软件研制能力的发展,对传统的、以人工为主的软件测试提出了挑战;基于DevOps思想,测试团队提出了一种航空机载软件测试工具链设计方案,实现了机载软件的自动化测试;开发人员提交代码后可以更快地得到反馈,软件错误也能更快地得到修复;同时,设计的交叉测试环境采用虚拟化和仿真的手段,使嵌入式软件能够在普通的电脑运行和测试,解决嵌入式软件测试硬件不足的问题;测试工具链在航空机载软件测试中得到了很好的应用。

关键词:测试工具链;测试自动化;交叉测试环境;DevOps

Design and Application of Airborne Software Testing Tool Chain

Li Chang, Jiang Youyi, Song Yanxiang, Feng Lei
(AVIC, Chengdu 610091, China)

Abstract: With the development of airborne software development capability, the traditional manual software testing has been challenged. Based on the DevOps idea, the test team proposed an aviation airborne software testing tool chain design, which realizes the automated testing of airborne software. Developers can get feedback faster after submitting code, and software errors can be fixed faster. At the same time, the cross-test environment designed can make embedded software run and be tested on ordinary computers by virtualization and simulation, and solve the problem of insufficient hardware for embedded software testing. The testing tool chain has been well applied in the airborne software testing.

Keywords: test tool chain; automated testing; cross test environment; DevOps

0 引言

随着航空飞行器研制发展,航空机载软件呈现出了数量多,代码量大,逻辑复杂等特点。航空机载软件的研发也采用了基于模型的系统工程方法,系统设计能力得到了较大提升;多个型号飞机并行研制增加了机载系统数量,需要同时开发、测试和维护的软件产品数量多、软件状态复杂。传统的软件测试的模式已经难以适应当前的航空机载软件的研发。如何提前发现问题,保障软件质量对传统模式下的软件测试提出了挑战。

1 背景

近年来,随着科学技术的迅速发展以及高新技术在新一代战斗机中的大规模运用,航空机载系统的能力不断提升,系统的综合化和复杂度也越来越高[1]。航空机载软件是实现飞机功能的重要载体,与以往三代飞机相比,新一代飞机除了软件规模及复杂度呈指数上升以外,机载软件的研制主要发生了以下两大变化:

首先,系统设计模式发生了变化。基于模型的系统工程(Model-Based Systems Engineering, MBSE)方法在型号中逐步推广及实践。MBSE主张以模型的形式支撑系统和软件设计,持续贯穿整个系统研制过程。MBSE的使用,软件设计人员基于系统模型,在完成软件需求分解、软件架构与功能设计后,代码可通过模型驱动的开发手段自动生成,大大提高了软件开发的效率。

其次,多个型号并行研制,需要并行开发、测试和维护的软件产品数量多、软件状态复杂。随着机载软件在飞机所占功能比例越来越大,重要度越来越高,软件从原来附属于成品设备,剥离出来作为产品来进行管理,相应的技术管理要求更为严格;企业在承担型号任务上,从原有一两型飞机研制向多型号并行研制转变,同时承担的有人飞机、无人飞机达到十多个,软件产品研制呈型号多、产品多、技术状态多的“三多”特点。

新的系统设计模式采用及软件多项目并行的现状,对现有以人工方式为主的机载软件测试提出了巨大的挑战,需要与之适配的软件测试手段。挑战主要包括以下两个方面:

首先,MBSE方法的引入,带来了测试工作的“左移”和“右移”,如图1所示。测试左移是指测试向软件研制的早中期移动,测试模式将向测试驱动开发(TDD)模式转变。在软件研制的早中期,软件版本发布频繁,需求变更范围大,要求版本迭代周期为1~2周完成一个版本的软件测试工作,对测试效率要求非常高;测试右移是指测试向软件研制的验收、维护阶段移动。系统及软件设计人员很重视软件验收测试的效果,要求测试人员对系统功能进行完整的功能覆盖测试。针对系统、外场试飞反馈的故障,要求测试人员能够对故障进行分析,对系统、软件问题快速复现和定位。

收稿日期:2018-12-13; **修回日期**:2019-01-05。

作者简介:李 昌(1983-),男,广西梧州人,大学本科,工程师,主要从事机载软件自动化测试方向的研究。

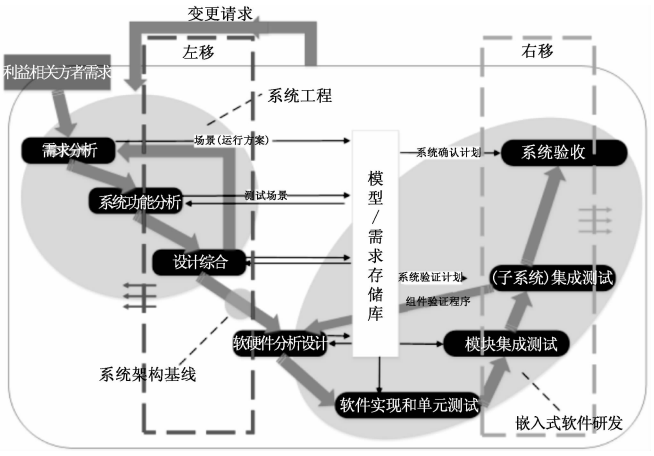

图 1 测试“左移”和“右移”

其次，由于软件多产品线并行，软件测试工作量十分巨大。在型号进度要求相近的情况下，各个软件需要同时开展软件测试，按照传统的手工执行测试的方式，存在测试人员瓶颈的问题。

近年来，DevOps 理念在互联网企业取得广泛的推广和应用。DevOps（Development 和 Operations 的组合词）可以看作开发、技术运营和质量保证三者的交集，是一种重视“软件开发人员（Dev）”和“IT 运维技术人员（Ops）”之间沟通合作的文化、运动或惯例。透过自动化“软件交付”和“架构变更”的流程，来使得构建、测试、发布软件能够更加地快捷、频繁和可靠[2]。DevOps 强调团队协作，自动化，虚拟化以及软件开发、测试及运营活动的新工具的桥接，目标是缩短反馈周期和开发周期，帮助软件企业通过频繁和自动化的软件发布来提高对需求变更的响应能力。

我们不难找到系统工程与 DevOps 之间的对应关系。开发（Dev）过程被描述为涉及需求分析、设计、实现以及系统集成和测试的一系列活动，而操作（Ops）过程通常与维护活动一起，主要集中在软件的部署及使用阶段。之所以没有单独把质量保证（QA）活动单独出来，并不是因为不重要，而是因为质量保证活动包括评审、测试、度量等活动，贯穿于整个软件过程的活动。

打破部门之间的“竖井”是提高团队协作的有效途径。大部分航空企业的软件开发、维护保障和质量保证部门为各自独立的部门，不同部门之间的目标是有差异的，比如，软件开发部门目标是尽快将变更后的软件版本部署到系统，它希望测试工作能够迅速完成；软件测试部门目标是排查软件隐藏的问题，它希望测试输入足够充分，并且需要预留充足的时间完成软件测试，保证测试的高覆盖率。当相互依赖的工作被分开时，会存在大量的信息延迟或者丢失，从而延长了整个软件研制周期。

自动化测试是持续集成思想的重要实践。持续集成是指一种软件工程流程，是将所有软件设计人员对于软件的工作副本持续集成到公用主线的一种方法[3]。通过频繁的自动化的构建（包括编译、发布、自动化测试）来，可以尽快地发现集成错误。比如，在测试驱动开发（TDD）的实践中，通常配置自动化的单元测试和代码运行时错误检查来发现软件缺陷。

要开展自动化测试，光靠持续集成框架和测试工具，也许在互联网企业是可行的，但并不适用于航空企业。有两方面原因：一是，机上软件大多数为嵌入式软件，难以脱离目标系统单独运行，需要有真实目标系统。但是航空企业都为研发驱动测试模式，软件测试部门无独立的硬件目标系统用于软件测试。二是，在自动化测试实践中，硬件难以支持设置、控制，要想完全测试自动化，需要对目标系统进行定制。芬兰奥卢大学 Lucy Ellen Lwakatare 在一项关于 DevOps 在嵌入式领域应用的研究指出，对硬件依赖性，以及缺乏在特定环境中自动、可靠和反复部署软件新功能的技术是阻碍 DevOps 在嵌入式领域应用的主要最大挑战[4]。

为突破 DevOps 在嵌入式领域的最大挑战，需引入全系统仿真技术。全系统仿真技术是对真实目标系统的虚拟化，解决了对测试硬件环境依赖的问题。全系统仿真系统是一套虚拟平台，能够完全仿真目标系统中对应的实际硬件[5]。开发、测试人员在这套虚拟平台上运行被测试软件，例如固件、操作系统、中间件或应用软件时，可以获得在实际目标系统上运行这些软件的效果。

商用、开源的软件开发、测试工具是软件测试工具链的执行模块。这里所说的工具链是指编译、构建、测试所使用到的开发集成环境和测试工具。通过对市场上面向嵌入式软件的商用、开源软件开发、测试工具（主要为国外工具）进行调研，主流的测试工具都提供了开发接口，能够满足与持续集成框架的二次开发集成的需要。

因此，为突破企业现有软件测试手段和人力资源的瓶颈，基于 DevOps 理念，从如何有效开展嵌入式软件自动化测试作为切入点，测试团队提出了一种基于 DevOps 的航空机载软件测试工具链设计方案。设计方案主要从业务、架构、技术三个方面展开阐述[6]。

在业务视角，提出了企业内敏捷、协同的软件测试流程。目标打破企业内软件开发、测试之间的“竖井”，通过该流程实现在开发、测试各个团队之间的密切合作。

在架构视角，提出了软件测试工具链的架构思路。

在技术视角，提出了持续集成环境、全系统仿真环境、测试工具环境的搭建方案。持续集成环境是测试流程中各个环节的链条。实现了软件每日构建（Nightly Build），以及自动化测试中待构建代码、测试用例、目标系统仿真模型库、测试执行结果的获取和推送的功能。全系统仿真环境是自动化测试的运行载体。实现了目标系统的模拟仿真，仿真目标系统的容器化及测试云化部署。测试工具环境是测试流程的执行单元。测试工具环境选择松耦合方式的方式进行设计，便于测试流水线按照项目进行个性化配置。测试工具环境的集成开发考虑了先进性、继承性、成本三

方面的因素，主要有三个做法：一是充分借鉴国外软件测试经验，引入优秀的开源工具。发挥开源工具灵活、可配置强的特点；二是整合企业已有的软件测试工具，使测试技术保持继承性；三是采购具有关键技术的商用工具，保证测试过程的稳定和可靠。

2　工具链设计

2.1　业务流程

航空机载软件的研制阶段分为 F（设计方案）、C（初样设计）、S（试样试制）、D（设计鉴定）、P（批型）阶段。软件测试主要集中在 C、S、D 阶段，各阶段测试业务需求如下：

在 C 阶段，软件需求、软件架构功能代码已经确定，硬件目标系统仍未设计完成。应采用测试驱动开发方式，尽早考虑，有利于保证后续自动化软件测试的开展。该阶段软件测试的主要工作参与软件需求的讨论，进行测试需求分解；根据测试需求，搭建软件单元、配置项自动化测试框架。

在 S 阶段，由于系统设计变更和软件功能设计更改，软件需求、代码变更十分频繁。该阶段的软件研制具有以下两个特点：一是机载系统状态批次增多，软件状态复杂，测试人员要清楚识别、区分软件版本状态；二是机载软件研制 S 阶段周期较长，软件开发以增量迭代的方式进行，待测试软件较上一版本变化量小，但累计发放版本多，每个版本均需要保证软件测试质量。针对 S 阶段的软件研制特点，该阶段软件测试的主要工作为建立测试流水线机制，一条测试流水线对应一项软件的测试环境配置，保证软件版本及测试受控；根据软件需求、设计文档，进行测试设计，使用测试流水线驱动自动化单元、配置项测试；根据软件变更单，进行软件变更影响域分析工作，进行回归测试。

在 D 阶段，该阶段软件已通过设计鉴定，文档、代码状态稳定。该阶段飞机各项科研试飞工作仍在继续，软件仍需排故、升级及维护保障[7]。该阶段软件测试为保证与真实环境一致性，以确认测试为主，主要在系统综合试验台进行；另外，全系统仿真环境配合进行补充功能验证，可帮助问题的定位、排查。

通过对软件研制阶段测试业务需求进行分析，机载软件测试的瓶颈在于两个地方：一是测试与开发协同的问题。即如何建立开发测试的协作机制，保证多技术状态、多版本软件测试情况下保证测试状态准确；二是测试效率的问题。即在测试人力资源相对有限的情况下，如何构建自动化测试框架，并利用自动化测试框架，提高测试效率，完成测试任务。

因此在考虑软件测试工具链架构设计时，需要考虑以上的因素。结合软件测试业务需求，工具链业务架构如图 2 所示。

在对软件测试业务流程分解后，得到业务、工具链功能、操作对象、工具集成技术的对应关系。如表 1 所示。

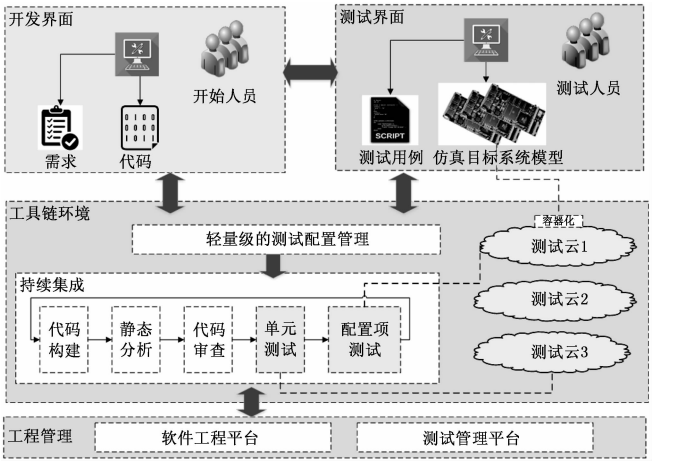

图 2　测试工具链业务架构

表 1　业务流程与工具链的对应关系

业务	工具链功能	操作对象	工具集成技术
提交代码、文档及获取代码、文档	开发、测试界面的配置管理，与配置管理适配的持续集成环境	代码、文档	分布式版本控制工具，配置管理远程仓库工具，持续集成流程引擎
获取测试模型、用例及过程数据	测试界面的配置管理，工具链界面的配置管理	测试模型、用例、过程数据	分布式版本控制工具、配置管理远程仓库工具
开展文档、代码审查	开发测试人员在统一工作界面进行文档、代码审查	文档、代码、软件缺陷	基于 B/S 架构的文档、代码审查工具
开展代码静态分析	代码编译、构建，代码运行时错误检查，代码规则检查，代码质量度量	代码	嵌入式开发 IDE 工具、运行时错误检查工具、代码规则检查工具、代码质量度量工具
开展单元测试设计	支持单元测试用例设计	设计文档、代码	支持断言的自动化单元测试框架
开展配置项测试设计	支持配置项测试用例设计	需求、设计、接口文档，代码	全系统仿真外设建模技术，Python 脚本
开展测试环境部署	支持全系统仿真系统模型在服务器的部署	仿真模型、目标码、操作系统镜像	容器技术、微服务技术
开展并行的、自动化的软件测试执行	支持多测试任务并发构建的自动化软件测试	测试用例、测试脚本、测试过程数据	持续集成流程引擎 批处理技术
测试数据、测试问题收集和展示	支持持续集成方式的数据自动收集、读取	软件缺陷、测试数据	缺陷跟踪工具，CVS、XML 集成、Python 爬虫技术

2.2 工具链架构

工具链定义：多个工具结合使用时，可以支持和构建领域工程的工作流。[8]结合企业内机载软件测试业务需求，对工具技术架构进行了规划，工具链构建应当求精而不追求面面俱到，设计准则归纳为 4 个字：“小”、“独”、“轻”、“松”。“小”是指粒度小，只关注某个任务的实现。在构建测试流水线时，要细分流水线的环节步骤。最好一个环节对应到一个工具；“独”是指每个微服务是单独的进程。减少甚至不做工具之间的集成，降低调用服务的复杂度；“轻”是指轻量级的通信机制，尽量使用 HTTP 接口。在不要求实时性前提下，减少中间件的使用，降低系统复杂性；“松”是指工具链层间、工具之间要松耦合，可独立部署。

基于“小、独、轻、松”的设计原则，将软件工具链设计为四层：服务、调度、集群及数据管理。测试工具链架构如图 3 所示。

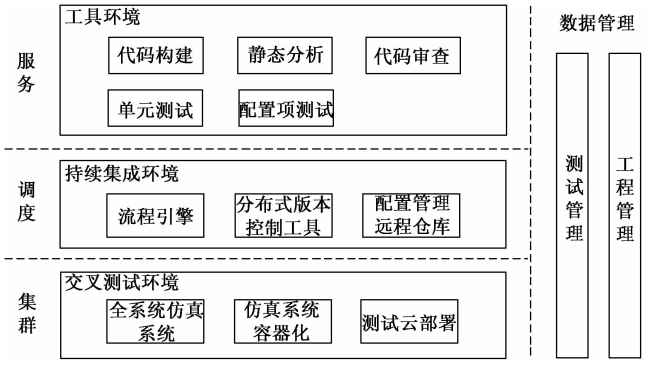

图 3 航空机载软件测试工具链架构

其中，服务层是将软件测试业务流程中使用的开发和测试工具环境，封装为服务的形式提供测试人员使用。包括代码构建、静态分析、代码审查、单元测试和配置项测试工具。

调度层内建一套驱动服务层、集群层的持续集成环境。调度层负责测试任务的创建、调度、管理，并根据测试任务需要，进行测试过程数据的自动抽取和推送。调度层包括流程引擎、分布式版本控制工具和配置管理远程仓库。

集群层是配置项测试的交叉测试环境。利用全系统仿真技术构建目标系统的仿真系统，负责测试脚本和目标代码的运行及结果输出。为解决调度层所创建测试任务自动化并发执行的问题，先使用容器技术对仿真系统封装为微服务，通过测试云部署将容器虚拟化部署。

数据管理层负责测试过程数据、测试问题的收集和展示。包括测试管理、工程管理两部分。包括测试管理和工程管理模块，负责提供接口将上述数据推送到企业已有的软件测试管理平台和工程管理平台。该部分将不作为本文描述的重点。

接下来，本节的其他内容将重点描述调度、集群、服务之间的关系。主要包括持续集成环境、交叉测试环境、工具环境。

2.3 持续集成环境

2.3.1 流程引擎

流程引擎是一个持续集成的框架，可以按照时间、事件方式驱动任意的 Shell 脚本和 Windows 批处理命令的执行。构建流程引擎的目标是为不同的被测软件建立专门的自动化测试流程，并且监控软件测试流程，便于快速发现问题，提高测试效率。

流程引擎主要包括作业构建、作业流水线、分布式构建、构建后操作 4 个部分的功能：第一，作业构建功能分为串行构建、并行构建两种方式，需要根据测试任务复杂度情况及效率要求决定采用哪一种方式来进行作业构建。一个测试任务由若干个测试作业组成，要考虑过程数据监控、构建后数据采集的难易程度，合理进行测试作业划分时颗粒度。比如，对于代码静态分析任务，可以创建一个测试作业来进行构建，因为代码可以作为整体来进行分析评估，获取分析结果；对于单元测试任务，如果只是创建一个测试作业就显得颗粒太粗，不便于测试过程出错的故障排查，因此需要按照软件模块、单元进行结构化划分，构建为许多个测试作业；第二，构建好的测试作业是孤立的、无方向的，需要流水线将测试作业进行串联。作业流水线功能提供了流程建模的功能，将代码提取、代码构建、代码分析、测试等一系列测试任务串联起来。通过流水线功能，可根据不同的测试要求，快速的将已有测试作业组合在一起，形成针对某个版本软件的自动化测试流程；第三，分布式构建主要目的是为了并发构建，提高测试执行效率。将服务器定义为主节点，其他的计算机定义为从节点，分布式构建功能定义了主节点与从节点协同构建的策略。也就是主节点根据每个从节点的忙闲状态，动态的将测试作业分配到从节点来执行，由从节点将执行结果返回；第四，构建后操作功能完成测试作业执行后所产生的数据的采集及存储。数据的采集是通过 Python 爬虫的方式实现，从结果文件中抽取数据，并进行数据整合，形成结构化的测试结果。

流程引擎与其他系统之间的交互关系如图 4 所示。

流程引擎使用分布式版本控制工具的 SSH 权限，配置测试作业对配置管理远程仓库的访问权限；流程引擎从配置管理远程仓库获取代码、测试模型，向远程仓库推送测试过程数据；测试引擎通过分布式构建功能，分配测试作业到交叉测试环境进行测试执行，从交叉测试环境读取测试结果；作业流水线执行完成后，通过构建后操作功能，推送测试结果到测试管理平台和工程管理平台。

前期对市场常用的流程引擎进行了调研，主流的流程引擎工具有 Jenkins、Teamcity、Travis CI、Bamboo 等。目前开源的流程引擎拥有广泛的社区基础，具有版本迭代快，功能稳定的特点，因此考虑开源工具进行流程引擎的集成

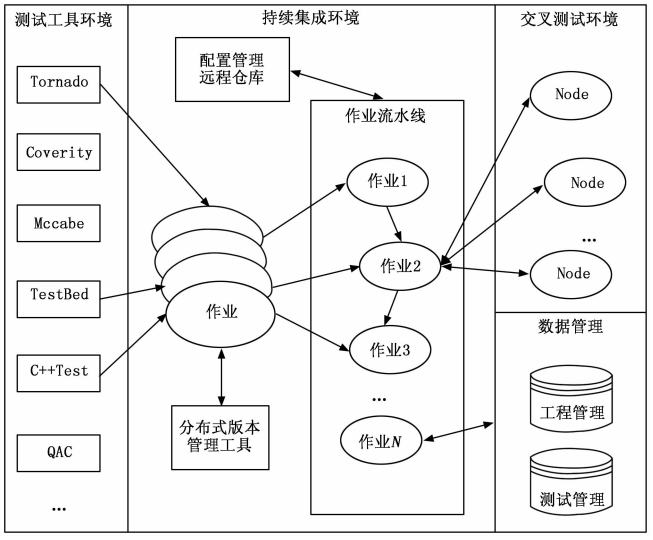

图 4　流程引擎与其他系统的交联关系

开发。

2.3.2　分布式版本控制工具

由于工具链涉及到软件开发人员、测试人员、工具之间的信息交互和协同，要求之间的交互是简单而敏捷，因此考虑轻量级的配置管理解决方案，使用分布式版本控制工具作为测试过程的配置管理工具。分布式版本控制工具与集中式版本控制工具的区别是分布式版本控制系统没有中央服务器，每台计算机都拥有一个完整的版本库。就算服务机宕机，也可以正常进行版本的变更和提交，不会影响当前的开发、测试工作。

以往企业内测试团队要同时解决测试用例版本之间、测试用例与代码版本，测试用例版本与环境配置之间对应的难题，这往往是十分繁琐且工作量巨大！分布式版本控制对测试团队非常有效，采用分支机制、集成管理者工作流和配置管理远程仓库，使得人、工具之间的协作变得高效。

目前世界上最主流的分布式配置管理工具是 Git，具有速度快，设计简洁，完全分布式设计的特点。因此考虑使用 Git 作为测试团队的版本控制工具。

2.3.3　配置管理远程仓库

配置管理远程仓库是代码、测试用例、测试模型及文档配置管理服务器软件。远程仓库在工具链中具有两个作用：1）测试版本控制协同。测试人员通过分布式版本控制工具，将测试数据提交到远程仓库，提交版本分支合并的变更请求；通过远程仓库获取最新的测试数据版本，在本地进行工作；2）作为流程引擎的输入和输出。流程引擎通过钩子机制，监听远程仓库的版本变更情况。如果发生版本变更，自动触发从远程仓库抽取数据，驱动测试作业的执行。业务流水线执行完毕后，通过构建后操作，将测试结果推送到远程仓库进行保存市面上的配置管理远程仓库软件比较多，主流的有 GitHub、GitLab，Bitbucket，Coding，这里面既有开源的自托管项目，也有商业软件。

2.4　交叉测试环境

交叉测试环境是指嵌入式软件在通用计算机环境（相对真实目标系统）中运行测试的环境。在本文所指的交叉测试环境，是用于支持航空机载配置项测试执行的全系统仿真系统、仿真系统部署环境（容器化环境和测试云）。

交叉测试环境在测试工具链的应用，是利用全系统仿真、软件容器、云技术，在软件配置项级开展并发式的功能测试及快速的回归测试，加快问题定位和排查故障的效率。

2.4.1　全系统仿真系统

全系统仿真系统是一套仿真嵌入式目标系统的虚拟平台，能够完全仿真目标系统中的 CPU、寄存器、内存、外设接口，全系统仿真系统示意图如图 5 所示。开发、测试人员在这套虚拟平台上运行被测软件，可以获得在实际目标系统上运行这些软件的效果。在全系统仿真系统进行软件测试，与实际目标系统相比，具有以下两个特点：1）支持检查点分析。通过插入检查点，全系统仿真系统把软件运行状态保存下来，支持运行状态回放分析，便于软件问题定位；2）支持访问目标系统状态。例如在真实硬件上，很难去跟踪控制寄存器、状态寄存器的内容，但通过全系统仿真系统，目标系统就像一个白盒，可以清楚看到内部的状态，这对软硬件问题的排查十分有帮助。[9]

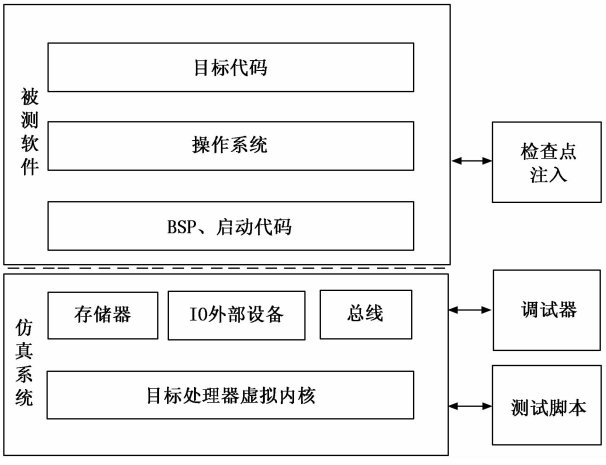

图 5　全系统仿真系统示意图

全系统仿真系统要在机载软件测试使用，除要有 CPU 仿真模型外，还需要进行外围设备仿真建模。外围设备仿真建模主要指对目标系统外围器件、接口的建模仿真，包含各种器件如内存、寄存器、外部定时器、FLASH 等，各种标准的通信接口，如 RS422、RS232、CAN、1553B、1394B 以及 FC 总线。

由于 CPU 仿真模型的构建难度大，这类工具厂家数量在国内外并不多。适合航空航天选型芯片的工具主要有美

国风河公司的 Simics 和上海创景公司的 VDVP 产品。其中 Simics 以支持 Power PC 芯片系列为主，VDVP 主要侧重 DSP 芯片系列。

2.4.2 仿真系统部署

使用软件容器技术和云技术进行仿真系统部署。

软件容器是一种应用程序的封装技术。软件容器在 Linux 操作系统上，提供了一个额外的软件抽象层以及操作系统虚拟化的自动管理机制，为应用程序的运行提供所需资源。[10]创建一个容器，将全系统仿真系统封装在容器中，可以实现全系统仿真系统从 Windows 向 Linux 的移植。软件容器对计算机资源占用较少，在同等配置的服务器上，可以保证性能的情况下，运行多个容器。

云技术是指在广域网或局域网内将硬件、软件、网络等系统资源统一，实现数据的计算、存储、处理和共享的一种托管技术。把交叉测试环境看作为提供测试服务的测试资源池，软件容器就是可共享的提供测试服务的资源。云技术负责提供软件容器的动态创建，应用程序装载，与流程引擎的调度机制。

基于软件容器、云技术，提出了一种仿真系统的部署方案。仿真系统部署方案如图 6 所示。

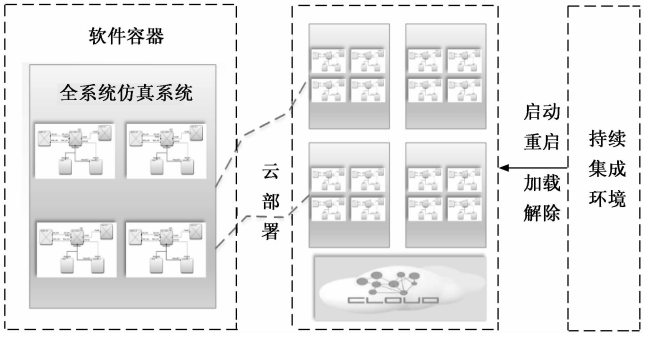

图 6 仿真系统部署方案

2.5 工具环境

工具环境是将软件测试业务流程中使用的开发和测试工具，封装为服务的形式提供测试人员使用。工具环境包括代码构建、静态分析、代码审查、单元测试和配置项测试工具。

工具环境的构建思路是整合现有的软件测试，将现有脱机使用的测试工具往服务器进行迁移，形成企业内部测试资源池；引入优秀的开源工具，特别是在代码级、单元级测试，往通用软件测试思路转变；开发、测试工具在与调度层进行集成时，要保持“独”和“松”的原则，开发、测试工具之间不做集成开发，通过流程引擎进行调度。

3 工具链应用

根据以上工具链设计方案，进行航空机载软件测试工具链的搭建。现已完成持续集成环境、静态分析环境、协同代码审查环境、单元测试自动化环境的搭建，并在某机电系统软件的软件测试中应用，达到了预期的效果，实现方案如图 7 所示；完成了基于 Linux 系统嵌入式软件的交叉测试环境的搭建工作，为后续机载软件交叉测试环境的搭建奠定了基础。

某机电系统软件测试工具链的实现技术如表 2 所示。

表 2 某机电系统软件测试工具链实现技术

功能模块	工具、集成技术
流程引擎	Jenkins 工具(开源)
分布式版本控制工具	Git 工具(开源)
配置管理远程仓库	GitLab 工具(开源)
开发工具	WindRiver Tornado 2.2 Lampda SVM
静态分析环境	SynopsysCoverity Prevent 工具、PRQA QAC 工具、LDRA TestBed 静态分析模块
代码审查环境	JetBrains 公司 Upsource(有限制免费)
单元测试	LDRA TbRun 工具 BeautifulSoup 4、lxml 爬虫(开源)
测试管理	企业内的测试管理平台
工程管理	软件工程管理平台

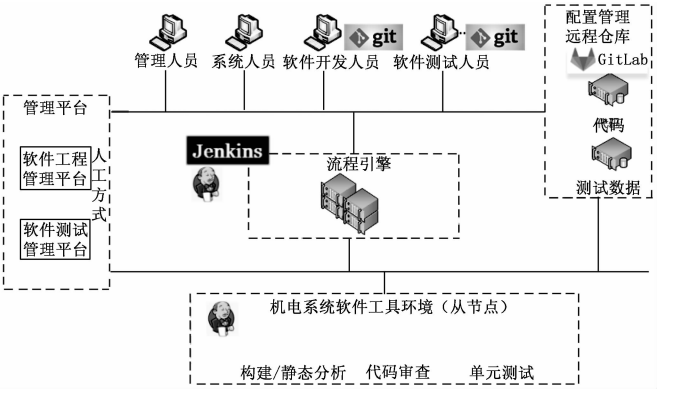

图 7 某型号机电系统软件工具链实现方案

3.1 持续集成环境构建

采用 Jenkins、Git 及 GitLab 方案搭建持续集成环境，界面如图 8 所示。其中 Jenkins 作为主节点部署，GitLab 作为配置管理远程仓库部署在从节点，测试工具部署在另外的从节点。

测试数据版本控制采用分支机制、集成管理者工作流进行管理，也就是将工作分支分为 master、develop 两条分支，测试人员使用 Git 克隆远程仓库的 master 分支版本到本地仓库，在完成变更后，提交到远程仓库的 develop 分支，并在 GitLab 中向项目组长提出合并 master 分支的请求，由项目组长审核完成后将合并后的变更推送到远程仓库的 master 分支。

3.2 静态分析

采用 Jenkins、Git、GitLab、Tornado、TestBed、Co-

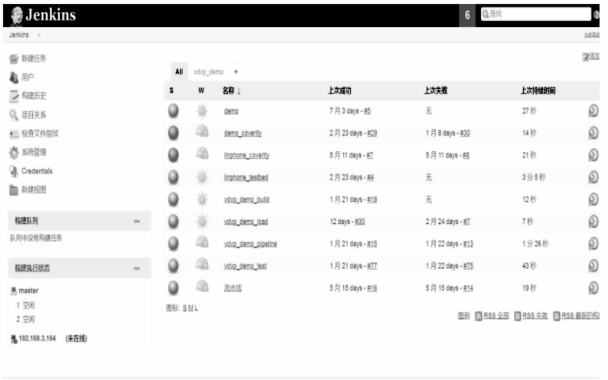

图 8　持续集成环境界面

vertiy 工具集成方案配置静态分析测试作业流水线，如图 9 所示。开发人员通过 Git 在本地仓库提交代码，流程引擎会根据测试作业流水线配置，自动化完成被测软件的自动提取、编译构建、运行时错误检查、代码规则检查、代码质量度量，并自动推送测试结果给开发测试人员查看，实现对软件的每日构建能力，如图 10 所示。

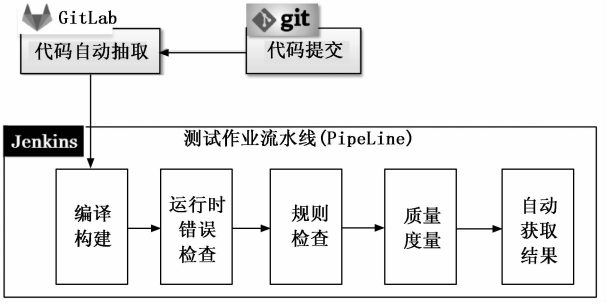

图 9　静态分析流程

Coverity Errors

ID	Checker	File	Function	Classification
1	COPY_PASTE_ERROR	/gitlab-proj/linphone/codecs/speex/libspeex/filters.c	qmf_decomp	Unclassified
2	FORWARD_NULL	/gitlab-proj/linphone/codecs/speex/libspeex/nb_celp.c	nb_decode	Unclassified
3	FORWARD_NULL	/gitlab-proj/linphone/codecs/speex/libspeex/nb_celp.c	nb_decode	Unclassified
4	FORWARD_NULL	/gitlab-proj/linphone/codecs/speex/libspeex/nb_celp.c	nb_decode	Unclassified
5	OVERRUN	/gitlab-proj/linphone/codecs/speex/libspeex/speex_header.c	speex_init_header	Unclassified
6	UNINIT	/gitlab-proj/linphone/codecs/speex/libspeex/filterbank.c	filterbank_psy_smooth	Unclassified
7	UNINIT	/gitlab-proj/linphone/codecs/speex/libspeex/sb_celp.c	sb_encode	Unclassified
8	UNINIT	/gitlab-proj/linphone/codecs/speex/libspeex/nb_celp.c	nb_encode	Unclassified
9	UNUSED_VALUE	/gitlab-proj/linphone/codecs/speex/libspeex/lsp.c	lpc_to_lsp	Unclassified

图 10　推送 Jenkins 的运行时错误检查报告

3.3　代码审查

采用 Git、GitLab、Upsource 工具集成方案配置代码审查协同工作环境。以往在项目中使用单机版的代码审查、比对工具（Source Insight、Beyond Compare）进行代码审查，这种方式存在两个问题：一是开发、测试人员缺少协同的审查环境。代码审查要求组织正式的评审会，以走读代码逻辑的方式来进行，但项目实际不允许这样的实践。二是代码与问题对应关系不直观。问题记录单与代码分开存放，时间长了关系不容易维护。通过代码审查协同工作环境，开发、测试人员可以约定在一定时间周期内开展审查工作，测试人员在线质疑代码中存在的问题，问题会自动推送给开发人员来进行解决。代码与问题的对应关系在系统中进行维护，十分方便版本之间的问题比较、分析。

4　结论

本文从软件测试角度作为切入点，研究航空机载嵌入式软件的结合方式，提出了一种航空机载软件测试工具链设计方案，完成了工具链技术架构的搭建，持续集成环境与静态分析、代码审查、单元测试工具环境的集成开发，并在某飞机的机电系统软件测试中进行应用，达到了提高测试效率的效果。测试人员在工具链的应用过程中，收集开发人员的反馈，作为测试工具链下一步优化的建议。

通过在企业内部开展航空机载软件测试工具链设计与应用的探索，深切感觉工具链的实质是让工具环境、自动化能力成为软件研制的一部分，以技术能力解决过去用人工加流程要解决的问题。但工具链只是人、机、料、法、环的其中一环，代表着一种新的思考方式和文化。相信随着国内机载软件研制能力的持续进步，更多的理念能够融入研制流程，提高软件研制水平！

参考文献：

[1] 蒲小勃. 现代航空电子系统与综合 [M]. 北京：航空工业出版社，2013.

[2] Wikipedia. DevOps [EB/OL]. (2018－04). https://en.wikipedia.org/wiki/DevOps.

[3] Wikipedia. Continuous integration [EB/OL]. (2016－07). https://en.wikipedia.org/wiki/Continuous_integration.

[4] Lwakatare L E, Karvonen T, Sauvola T, et al. Towards devops in the embedded systems domain: Why is it so hard [A]. 2016 49th Hawaii International Conference on System Sciences (HICSS) [C]. 2016: 5437－5446.

[5] Engblom J. Continuous integration for embedded systems using simulation [A]. Embedded world Conference 2015 [C]. 2015: 1－7.

[6] IEEE. (2000). IEEE 1471: 2000—Recommended practice for architectural description of software intensive systems [M]. Los Alamitos, CA: IEEE.

[7] 李　昌，邓矢斧，冯　雷，等. 基于全数字的航空机载软件验证平台研究 [J]. 计算机测量与控制，2018，26 (6)：130－133.

[8] Lu J, Chen D, Gürdür D, et al. An Investigation of Functionalities of Future Tool－chain for Aerospace Industry [A]. INCOSE International Symposium [C]. 2017, 27: 1408－1422.

[9] Engblom J, Ekblom D. Simics: A commercially proven full－system simulation framework [A]. Workshop on Simulation in European Space Programmes [C]. 2006.

[10] Wikipedia. Docker [EB/OL]. (2018－08). https://en.wikipedia.org/wiki/Docker_(software).