测试系统构建实用指南

软件部署

目录

引言

尽管设备日益复杂化，测试工程师却通常需要在日益紧迫的时间期限和缩减的预算下创建更复杂、更高级的混合测试系统。创建这些测试系统的一个最重要步骤是将测试系统软件部署到目标机器。这也是最繁琐、最令人沮丧的一个步骤。如今，对于那些只是简单寻求最便宜和最快速解决方案的工程师，琳琅满目的部署方法更加让他们无从选择。此外，测试系统开发人员也面临着其系统特有的各种考虑因素和敏感事项。

在本指南中，我们将部署定义为编译一系列软件组件并为将这些组件从开发计算机导出到目标机器上执行的过程。测试工程师不直接在开发环境运行测试系统软件，反而采用部署方法，主要是因为成本、性能、可移植性和保护等原因。以下几个示例是测试工程师需要从执行开发环境转变为二进制部署的常见情况：

- 每个测试系统的应用软件开发许可证成本开始超出预算限制。每个系统使用部署许可证是一种更有吸引力且更加高效的解决方案。
- 由于内存限制或依赖关系问题，测试系统的源代码变得难以传输。
- 测试系统开发人员不希望终端用户能够编辑或接触系统的源代码。
- 通过开发环境运行时，测试系统的执行速度会变慢或内存消耗增加。编译执行代码可提高性能，减小占用的内存。

本指南建议并比较了不同的考量因素和工具，以解决测试系统部署过程中的困难和困惑。尽管本指南可以解决测试系统部署的多个不同主题，例如源代码控制最佳做法或安装程序创建，但所选的主题应涵盖大多数通用部署问题。每个部分的结尾都提供了一个基本用例和高级用例的最佳范例建议：

- 基本用例是一个由可执行程序组成的简单测试系统，可按顺序运行测试步骤，并调用几个硬件驱动程序。这种类型的系统包含的测试功能通常少于200个。
- 每个基本用例的最佳实践最后都会阐述可能需要考虑高级用例的情况。

高级用例是指大型生产测试系统，使用可执行文件、模块、驱动程序、Web服务或第三方应用程序的组合来执行高度混合的测试序列。这种类型的系统通常包含数百或甚至数千个测试功能。

管理和确定系统组件

组件定义

在软件开发中，组件是指系统中使用的任何物理信息，例如二进制可执行文件、数据库表、文档、库或驱动程序。完成成功部署的第一步是确定与测试系统相关的组件，并确保每个组件都有相应的部署方法。该步骤可以非常简单，也可非常复杂。例如，简单测试系统的组件可能是单个可执行程序和必要的硬件驱动程序。

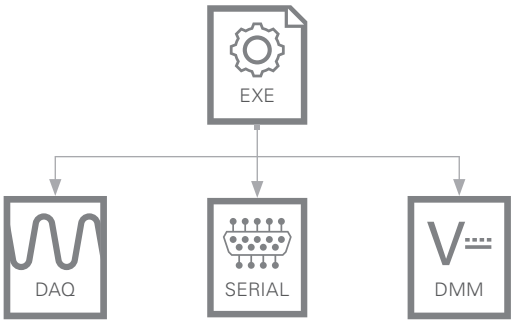

图1. 简单的测试系统可执行程序示例，包含DAQ、串行和DMM驱动程序

复杂的系统组件

然而，在复杂的测试系统中，这些组件通常是XML配置文件、数据库表、自述文本文件或Web服务。 较复杂的系统提供了更高级的部署选项。 例如，配置文件需要频繁地更新，以便针对季节性天气变化对采集的数据进行校准，而主要可执行程序很少需要更新。 每次配置文件需要更新时都不必重新部署可执行程序，因此配置文件可以使用与可执行程序不同的部署方法。

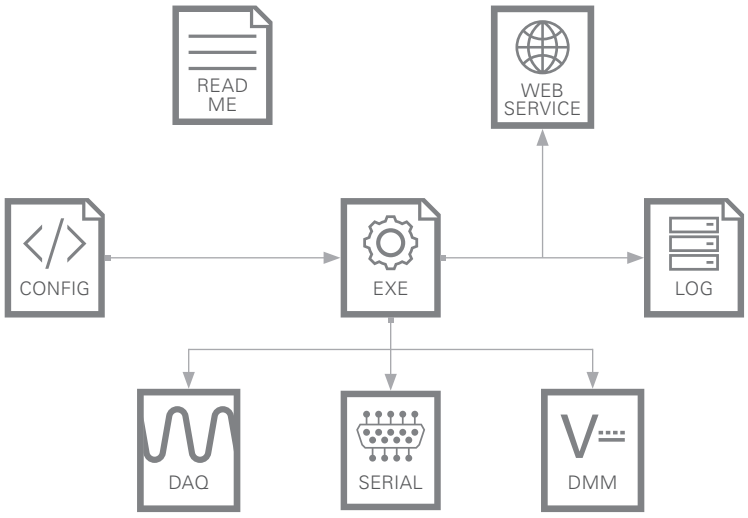

图2. 具有复杂依赖关系的测试系统示例

除了确定每个系统组件和设计其部署方法之外，确定系统组件之间的关系并确保部署方法不会中断这些关系是非常重要的。对于需要频繁更新的配置文件，工程师可能必须将配置文件安装到每个部署系统上的相同位置，以便可执行程序在运行时找到它。

依赖关系跟踪

维护依赖项之间的关系涉及安装一个依赖关系跟踪程序，以确保每个组件的依赖关系组件均已部署。 虽然在手动确定每个系统组件之后这一点看起来很明显，但是依赖关系通常可以深度嵌套，而且当系统扩展时需要能够自动识别。 例如，如果系统B中的可执行程序依赖于一个.dll文件来正确执行，制定部署计划的工程师可能忘记将.dll文件标识为必要组件或者不知道这个依赖关系。 在这些情况下，开发工具就派上用场了，可自动识别所编译的应用程序的大部分（即使不是全部）依赖关系。

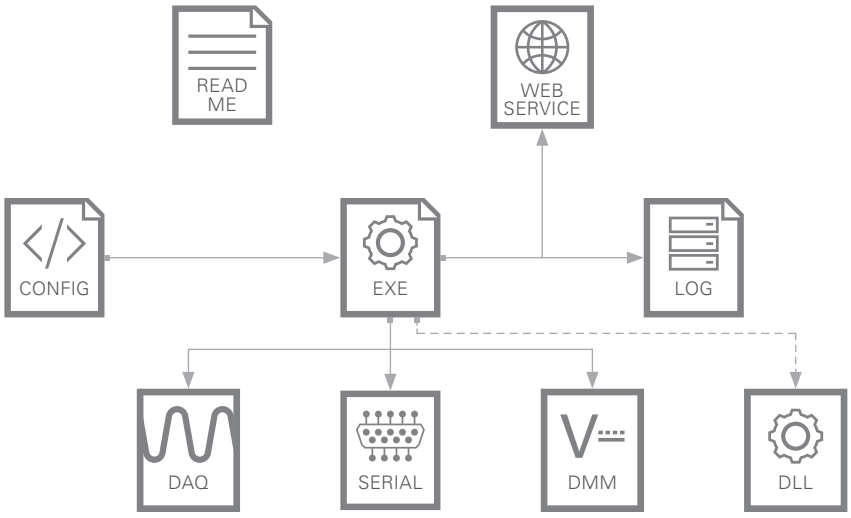

图 3. 复杂测试系统出现意外依赖关系

以下是几个开发软件应用程序示例：

- **LabVIEW Application Builder—**识别特定上层VI的依赖关系（子VI），并在编译的应用程序中纳入这些子VI
- **TestStand Deployment Utility (TSDU)—**以TestStand工作区文件或路径为输入，并识别系统的依赖关系代码模块; 自动编译这些模块并将模块纳入到编译的安装程序中
- **ClickOnce—**可帮助开发人员轻松地为其.NET应用程序创建安装程序、应用程序甚至Web服务的一项微软技术；可以配置为将安装程序纳入依赖关系，或者提示用户在部署后安装依赖关系
- **JarAnalyzer—**Java应用程序的依赖关系管理工具；可以遍历目录、解析该目录中的每个jar文件，并确定它们之间的依赖关系

关系管理

通常情况下，关系不仅存在于主测试程序可执行程序及其相关组件之间，而且存在于每个单独组件之间。 这使得不同组件或软件模块之间的关系的性质成为一个问题。 随着系统不断扩展，解析不同库、驱动程序或文件之间的依赖关系可能变得非常复杂。 例如，测试系统可以使用具有以下关系的三个不同代码库，如下图所示。

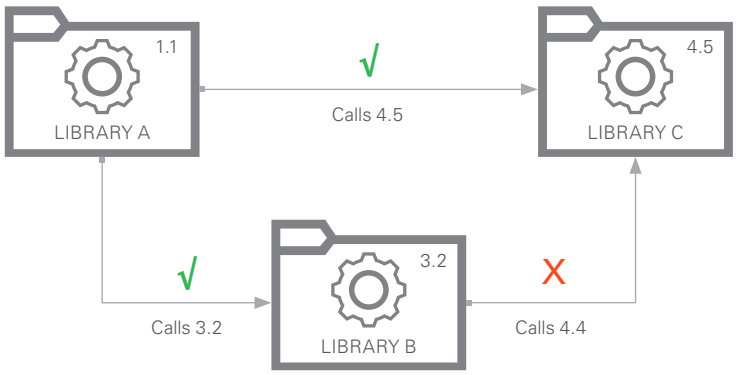

图4. 由于库A依赖于库C版本4.5，因而库B对库C版本4.4的依赖导致了一个不可解决的依赖性问题。

对于这些复杂的系统，通常需要使用依赖解析器来识别依赖冲突以及管理不可解决的问题。虽然我们可以自行编写依赖解析器，但工程师可以使用程序包管理系统来管理依赖关系。程序包管理器的一个例子就是NuGet，这是一个专为.NET框架程序包设计的免费开源程序包管理器。 另一个例子是用于LabVIEW软件的VI程序包管理器，可允许用户发布代码库，并通过API提供自定义代码库管理工具。

最佳实践

基本用例：对于基本或简单的系统，通常可以手动跟踪所有必需的组件。 使用软件应用程序或程序包管理器来管理依赖性并不是很必要，而且需要的前期成本过高。 然而，如果碰到警告信号，例如一直碰到依赖关系缺失问题或依赖关系不断增多，通常意味着需要更高级的依赖关系管理。

高级用例: 采用可扩展的依赖关系管理系统可让复杂的系统变得更易于维护和升级。 这可能意味着使用程序包管理器来诊断程序包之间或程序包与软件应用程序之间的关系，以了解和识别各种组件的依赖关系，维护这样的系统对于长期成功至关重要。

硬件检测

硬件断言

需要特定硬件设置的测试系统需要确定该硬件存在于系统上，并且当硬件不包含在部署计划中或不兼容时可执行应急计划。 虽然开发人员通常通过视觉检查测试机器并将硬件组件与原始开发系统匹配来手动完成硬件断言，但是更好的做法是假定测试系统是为第三方创建的。 测试系统的客户如何知道是否有不兼容的硬件？ 如果插槽或端口不正确，系统能否适应正确的模块？ 当硬件丢失时，系统是否可以解决或进行调整？ 在开发初期回答这些问题可以简化测试系统的扩展和发布。

硬件标准化

硬件断言的最终目标是确定预期系统和实际物理硬件系统之间完全一样。 为此，通常最有效的方法是先将每个测试系统上要用的硬件组件标准化：

- **文档描述** - 标准硬件中的组件列表应该可供每个新系统访问。 文档描述应该包含有关提供商、产品编号、订单号、组件数量、可更换组件、保修、支持政策、产品生命周期等信息。
- **可维护** - 硬件标准化最困难的问题之一是确保每个测试系统使用的硬件组件在将来仍然可用。通常旧硬件是指制造商标示为寿命终止(EOL)，并且需要刷新测试系统硬件的标准组件。从硬件升级和测试系统停机的角度来看，这种刷新通常非常昂贵。与硬件制造商共同探讨硬件组件的生命周期策略有助于减少未来面临的挑战。大多数硬件制造商，比如NI，提供生命周期咨询以及每个硬件组件生命周期的逐步终结。
- **可复制** – 应考虑是否需要全球性甚至区域性地部署硬件。确保有合适的硬件部署方法，以便在偏远位置快速搭建新系统。对于许多系统，维护备用硬件组件以便进行维护或紧急替换也是非常重要的。

开机自检(POST)

即使测试系统的硬件正确且正确连接，也必须对硬件进行简单测试，以确保系统在运行时其性能与预期一致。 幸运的是，大多数硬件组件都具有制造商配置的自检功能，以对设备的通道、端口和内部电路板进行简单检查。 在为每个测试系统供电时，应对所有连接的设备进行自检，以便在早期检查出故障硬件。 例如，每个NI设备都具有自检功能，可以通过设备的驱动程序API以编程方式进行调用。 为测试系统供电的第一步可以是调用每个设备的自检功能，并向操作员警告任何故障硬件。

别名配置

不幸的是，对硬件进行标准化不能完全确保相同的配置。 通常，需要使用硬件配置软件（例如Measurement & Automation Explorer (MAX)）来将硬件设备重新映射到别名。 例如，在安装所有硬件组件并为系统供电后，工程师可以使用MAX来检测系统上存在的NI硬件，并使用Windows设备管理器来查找非NI硬件。之后就可以通过编辑.ini配置文件来将硬件设备正确映射到别名。下图显示了此过程的可能输出。

别名	设备名称
PXI NI-4139	PXI 1 插槽1
PXI NI-3245	PXI 1 插槽2
PXI NI-2239	PXI 2 插槽1

表1.用于将物理硬件映射到测试系统别名的hw_config.ini文件

程序配置

LabVIEW软件中用于NI硬件的System Configuration API等库可以以编程方式生成所有可用实时硬件的列表并进行别名映射配置。 例如，测试系统可执行程序可以调用System Configuration API的查找硬件功能来生成可用NI硬件列表。 在该列表中，每个设备的别名属性可以通过硬件节点设置为预定义的名称。 这可能会导致系统出现问题，例如硬件设备映射到不正确的别名。 因此，工程师应将其与另一种保护措施（如手动确认映射列表或标准化硬件设置）结合使用。

最佳实践

基本做法：对于基本或简单的系统，重要的是确保预期的硬件存在于系统上。 将硬件标准化对于所有系统都是不错的做法，尤其是随着硬件系统数量的增加，硬件标准化尤为重要。 正确执行测试系统所需的机箱、模块和外设应进行文档记述并定期更新。 然而，验证正确的设备是否存在于系统中通常可以使用MAX之类的工具进行手动检查，而不是使用程序化或可重配置解决方案。 随着硬件系统随着模块和设备数量的增加而扩展，可能需要采用更高级的解决方案来防止硬件丢失问题。

高级做法： 对于复杂的系统，跟踪哪些硬件对于系统是必要的或者哪些硬件存在于系统中应该使用多种解决方案的组合。 与上述基本做法一样，不同系统的硬件应该进行标准化并进行文档记录。 如果要检测故障硬件，应通过开机自检(POST)来确保所连接的硬件能正常工作。 此外，当硬件标准化失败时，应当使用程序化或手动别名映射系统来自动将期望的设备重新映射到系统的别名。

依赖关系解析

依赖关系断言

我们通常需要制定一个计划来解决对部署系统的现有依赖关系和缺失依赖关系。 通常，部署的测试机器已经安装了测试系统镜像的部分依赖关系。 对于较小型的系统，我们可以简单地重新安装所有依赖关系来确保它们的存在。 然而，对于较大型的系统，我们需要先检查系统上是否存在这些依赖关系，这样有助于避免重新安装所有依赖关系。 这种做法称为依赖关系断言，有助于减少部署时间，但前提是要为不同的依赖关系制定计划。 “组件化 ”部分将进一步讨论如何通过组件化实现更快速的部署。

例如，测试系统可兼容14.0和15.0版本的NI-DAQmx驱动程序。 虽然测试系统可能要求安装NI-DAQmx 15.0，但它可能允许14.0版本作为依赖项。 尽管可以兼容15.0版本，但允许14.0版本而非15.0版本可能会改变测试系统的行为，比如跳过某些测试步骤或调用不同的函数。 所有这些更改都需要记录和测试。

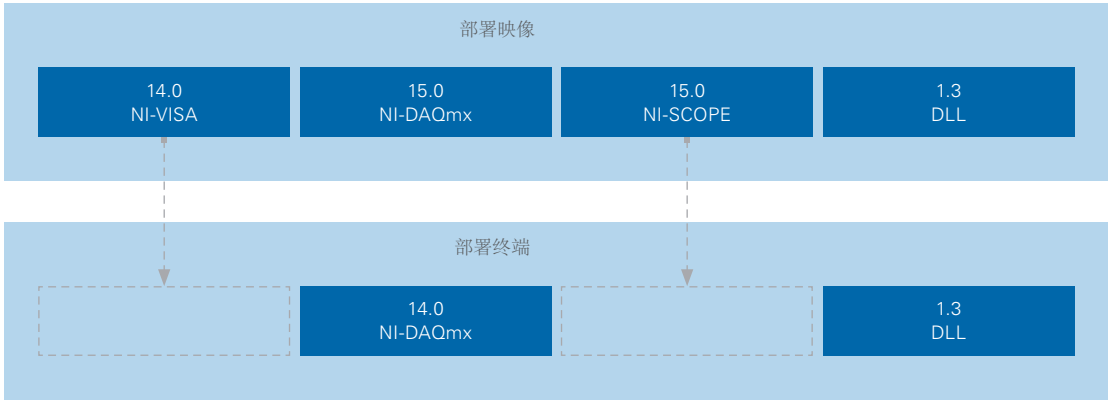

图5. 依赖关系断言

依赖关系断言的第二个要素是决定如何处理丢失的依赖关系。 如前所述，一个较好的做法是操作的时候想象测试系统部署到客户机器上的场景。 进行部署的工程师是否应该被通知缺少的依赖关系？ 缺少的依赖关系是否应该在后台悄无声息地安装还是应该由用户手动找到并安装？在初期回答这些问题有助于实现更快速的部署以及对缺失的依赖关系进行适当的处理。

最佳实践：

基本做法：对于基本测试系统，依赖关系解析和断言通常是不必要的。 安装所有测试系统的依赖关系（无论它们是否存在于系统中）通常比尝试识别并仅安装缺失的依赖关系更简单。随着测试系统的扩展，系统总安装时间可能会不断增加，增加到了一定程度，就需要开发依赖关系断言和解析工具。

高级做法：即使具有稳定的网络连接或采用压缩图像，部署时间也有可能会迅速增加到不合理的程度。 对于大多数高级测试系统，需要一定数量的依赖关系断言来避免重新安装所有组件。部署过程可以集成System Configuration API等用于查找系统中安装的NI软件的工具或用于生成Windows机器上所有程序的列表的wmic命令集。 这可以允许安装程序跳过特定组件或允许版本差异。

版本管理

通常，工程师需要知道当前部署到测试系统的软件镜像是哪个版本，或者能够提供发布部署历史。 如果这些是必要的需求，应该有一个版本管理系统来解决以下每个问题：

- 哪个版本目前部署到系统A？
- 系统B的最近部署状态是什么？
- 版本1、2和3部署到哪里？
- 系统A的发布历史是什么？

在大多数测试环境中，工程师使用铅笔和剪贴板系统来回答这些问题，但是，也有一些工具可以自动记录版本指标，并提供特定系统的版本历史记录。这些用于版本管理的工具可以集成到集成开发环境（IDE）中，或作为独立的版本管理工具存在。 其中一些例子包括：

- **Visual Studio Release Management—**Visual Studio IDE附带了用于自动化部署、版本历史跟踪和版本安全性管理的工具
- **Jenkins Release Plugin—**借助这个专用于Jenkins持续集成(CI)服务的插件，开发人员可以指定编译前和编译后操作来管理Jenkins集成开发的版本。
- **XL Deploy—**此应用程序版本自动化(ARA)软件可扩展到企业级，并提供可视化状态仪表板、安全性和分析功能来管理版本。

虽然上述几个工具可作为IDE和独立部署解决方案，但版本管理工具更经常与CI服务器和端到端部署过程结合使用。 这一点非常容易理解，因为我们更经常会问某个机器使用的是哪些代码，而不是哪个版本。 对于编译的系统镜像，这可能难以通过手动观察来确定。 跟踪从开发到部署的代码对于有效的版本管理是必要的。

端到端系统自动化

高效的版本管理是为测试系统部署开发复杂的端到端过程的必要组件。 从开发到部署，每个过程都依赖前一个版本；如果源代码管理良好，就能够很好地管理测试和版本。 有了良好的测试和版本处理，版本管理就只是原系统的简单扩展。 下图显示了一个典型的端到端系统。

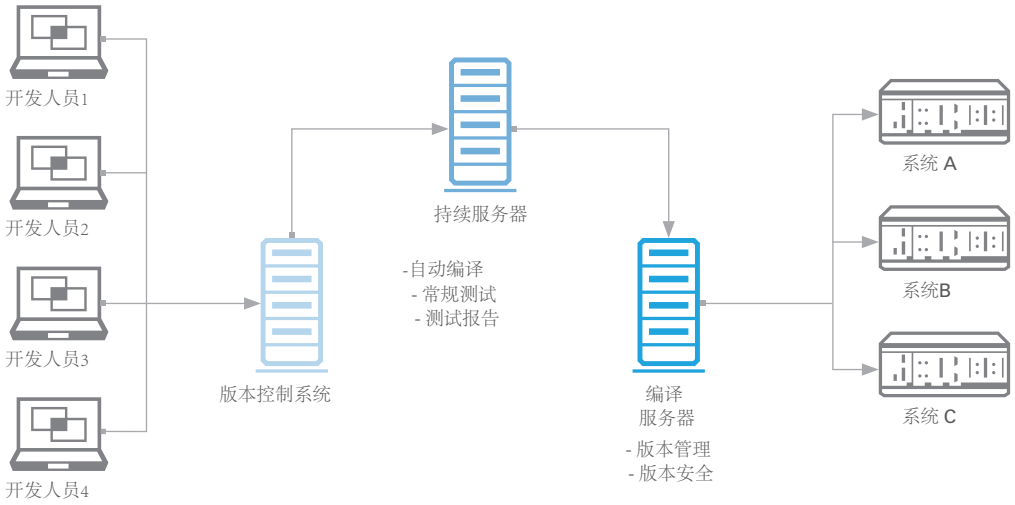

图6.开发人员将代码提交到版本控制存储库，然后可以CI服务器中进行编译和测试。 在服务器中，版本可以存储在编译服务器中并进行管理。

在此架构中，测试系统开发人员定期开发源代码并将其提交到版本控制存储库。 接着CI服务会将源代码拉入其自己的存储库，并相应地编译和测试代码。 此时，无论是自动还是手动执行，开发人员都可以将通过CI测试的版本移动并存储到编译服务器或存储库中。编译服务器通过报告和跟踪每个软件版本与特定测试机器的链接来进行版本管理。 通常，测试机器通过请求安装特定版本的测试系统来启动部署过程；然而，开发人员还可以配置编译服务器，以将镜像推送到所选择的机器上。

即使是在基本系统需要一定程度版本管理的情下，最实际的解决方案应反映版本需求的固有复杂性。 如果需求是跟踪哪个版本部署到系统上，则通过配置文件或对作为编译可执行文件的组件进行手动版本管理可能就足够了。 如果需求范围扩大，测试系统数量增加，或者应用程序版本数量增加，则必须对版本管理系统进行定义。

最佳实践:

高级做法: 通常，需要版本管理的复杂测试系统最适合采用一定形式的端到端自动化。 这可以通过CI服务（如Jenkins或Bamboo）来完成，这些服务会将版本管理与版本测试和源代码控制相结合。

版本测试

回归测试

在软件工程中，回归测试是指在对系统进行改动之后测试先前开发的系统的过程。 回归测试的目的是保持每个版本的完整性，并跟踪系统中特定更新或修补程序的错误。 对于组件化系统，回归测试对于确定升级到模块A是否会导致模块B中出现意外行为尤其重要。例如，升级系统中的NI-DAQmx硬件驱动程序可能会导致硬件抽象库在调用较旧NI-DAQmx版本但被较新版本弃用的函数时会出现问题。回归测试有两种类型：功能测试和单元测试。

功能测试

在测试系统中，最重要的问题是软件进行了哪些更新？这些更新是否会破坏系统的功能？系统是否仍然按照预期的方式运行？功能测试是通过一组已知输入验证系统是否会产生预期输出，可以帮助回答关于整个系统的这些问题。这种类型的测试通常采用“黑箱”方法；不对系统的内部机制进行分析，仅仅验证系统的输出是否与预期一致。对于测试系统，功能测试可能是验证硬件配置更新、驱动程序更改或测试步骤添加不会更改原始测试功能。工程师可以在模拟待测设备(DUT)的测试系统上执行功能测试，验证这些系统经校准后是否通过特定测试。例如，用于测试某个对象是否是圆形的系统由四个部分组成：相机控制器、圆周传感器、直径传感器和体积传感器。如果系统从版本1.0更新到1.1，并且对直径传感器进行了更改，则在下面的图中，正在测试的第二个圆形一开始会通过圆形测试器的测试，然后在系统更新后无法通过测试。

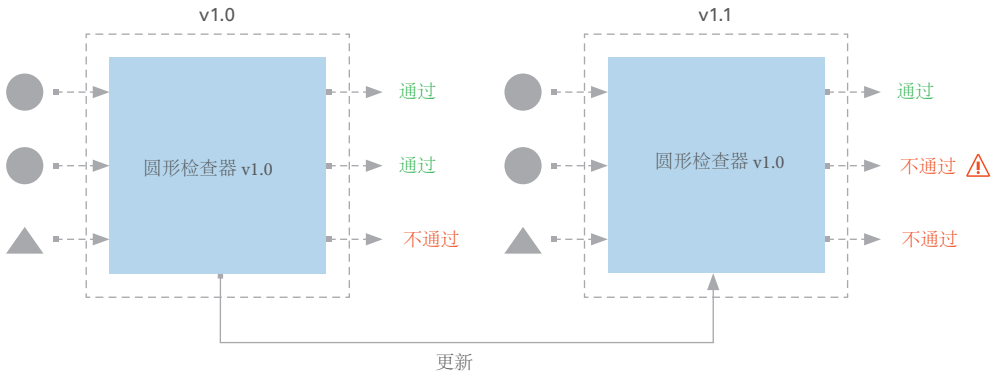

图7.对测试系统中的模块的小更新可能导致功能测试发生故障，这是一种伪故障。

单元测试

功能测试是针对整个系统，而单元测试则是针对特定的模块、组件或功能。这种类型的测试旨在跟踪测试系统特定部分的质量，而不仅仅是正确性。例如，测试结果记录到数据库后，可以在数据库控制器上进行单元测试以测量数据吞吐量。这样，不仅可以对数据库控制器的任何更改进行分析来对功能进行适当的记录，而且还可以回答软件更改是加快还是减慢系统的记录能力这一问题。除了帮助发现错误，单元测试可以将观察到的性能增强或降低与特定更改相关联。前面介绍的圆形测试器示例可以用来说明单元测试和功能测试之间的差异。假设圆形测试器的直径传感器软件部件进行了升级，则可以对直径传感器进行单元测试，而不需要对整个系统进行功能测试。对于单元测试，可以向特定组件提供表示具有特定直径的圆的二进制图像数据，并且测试输出是否与圆的已知直径匹配。通过这种方式，就可以验证和定量测量模块的正确性，例如测量模块的执行时间。

在这种特定情况下，升级显着降低了模块的速度。另外，由于升级后系统未通过功能测试但通过单元测试，我们还可以推断，软件漏洞很可能存在于相机控制器和直径传感器之间的通信中。 这种验证系统正确性和单个模块功能的能力可以确保只有合格的版本才会部署到测试机器上。

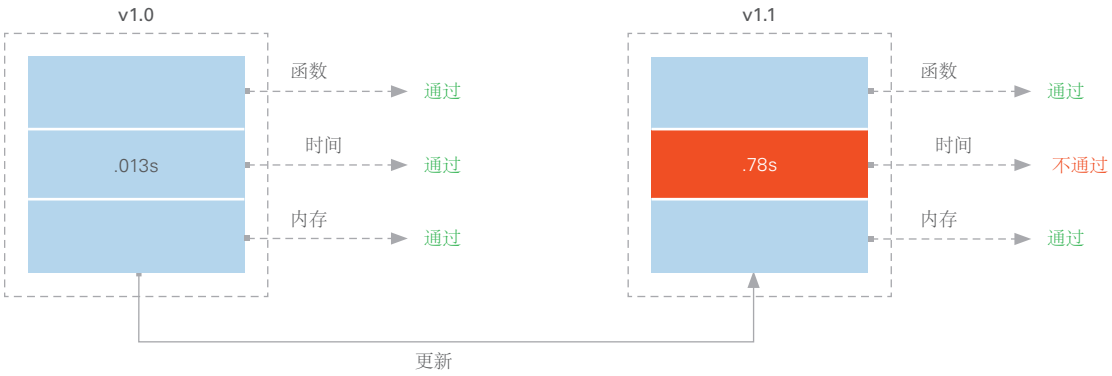

图8.对v1.0和v1.1执行单元测试后，更新后的处理时间被判定为一个问题，导致过程功能测试出现伪故障。

测试过程

为了节省开发时间，大多数测试系统中的回归测试与源代码控制、编译或版本管理同时进行。 这可允许重用测试代码，从而需要更频繁的更新。 然而，计划测试代码编译所需的开发时间和预算也很重要。 通常，回归测试是CI服务或IDE的组件，其中源代码控制、编译和测试按顺序执行。

最佳实践：

高级做法：对于所有测试系统，在将系统部署到新机器之前，应该进行一定程度的功能测试。功能测试的范围从使用模拟硬件在开发环境中手动运行应用程序等简单场景到基于配置文件运行一系列功能测试的复杂场景。 单元测试对于简单、单一的应用可能不必要，但是随着测试系统复杂性的增加，可能就需要进行单元测试。 随着添加的模块越来越多，特定的自定义测试对于跟踪漏洞或确保系统满足某些规格是必要的。

最佳实践：

高级做法: 复杂的测试系统不仅应该对每个新版本的测试系统进行各种输入功能测试，而且还要为系统的每个单独模块开发单元测试。 这两种回归测试方法都应该在部署过程中最有效的时间点上完成。 例如，功能测试应该在编译每个版本后进行，单元测试应该在每个源代码控制提交点执行。 通常，这些测试对于系统是强制性的或无需说明的，特别是在航空航天和国防工业中。

组件化

因为部署时间太长是大型测试系统的一个常见问题，所以最佳的做法是只更新测试系统中需要更改的单个组件而不是重新构建整个系统。 本指南的“依赖关系解析”一节部分解决了这一问题，但我们仍需要单独讨论如何开发模块化架构或基于插件的架构来实现更高效的部署。 无论工程师选择哪种架构，最佳做法都是定期更新外设模块，而更核心的模块则保持相对恒定，无需重新编译。 这种做法自然会产生有关更新频率的问题，稍后将在本节中探讨。

部署插件架构

对于软件部署，插件是指代码模块，其安装独立于主应用程序的安装，在功能上独立于其他插件，遵守全局插件接口，并且当用于编译的应用程序时可避免名称冲突。 主应用程序应该能够动态加载每个插件，通过标准接口调用每个插件，并使用每个插件作为扩展，而不需要重新编译。 成功开发后，插件框架可允许组件化部署、更新或安装特定或缺少的插件，而不需要重新编译主应用程序或任何未受影响的插件。

例如，为简单应用程序开发的插件框架可能包含主要可执行文件，会在加载时搜索插件目录，或在运行时执行定期搜索，并通过标准接口执行该插件。这样，插件可以连续部署到系统的插件目录中，而无需编辑主应用程序。

硬盘驱动器复制

通常，代码库、硬件驱动程序或特定文件是测试系统核心的一部分，不需要像其他模块化外设组件一样频繁更新。 在这些情况下，硬盘驱动器复制是将环境标准化为后续开发基准的一个好方法。 工程师可以将开发机器或初始测试机器的硬盘驱动器复制并克隆到其他测试机器上。 复制驱动器后，测试机器便拥有一个共同的起点，通常包括主测试应用程序或程序、必要的硬件驱动程序、系统驱动程序集和关键外设应用程序，如MAX硬件配置。 然而，重要的是要认识到，硬盘驱动器复制也有相应的注意事项，如测试机器之间需要相同的计算机硬件，或者占用大量内存的镜像，这使其不适用于软件频繁更新的情况。

使用硬盘驱动器复制为进一步测试开发奠定基础的一个例子是使用Symantec Ghost（一种常见的硬盘驱动器复制工具）和TestStand Deployment实用程序(TSDU)。在下图(A)的第一帧中，开发机器将其核心软件堆栈（红色）复制到目标机器上。这个核心软件堆栈是Windows操作系统、硬件设备驱动程序、运行引擎和MAX的组合。目标机器生成镜像之后，便可在开发机器上进行开发(B)，通过使用TestStand和LabVIEW（绿色）创建测试序列。然后，开发人员可以使用TSDU将测试序列移动到目标机器。对于测试序列的频繁更新，开发人员可以继续使用TSDU来节省开发时间，因为核心软件堆栈不需要更改。有时，开发可能是在未部署到目标机器(C)的开发机器上进行。系统不匹配可能会导致依赖关系缺失的问题。在这种情况下，开发人员可以选择重新为开发机器创建镜像，并将其复制到目标机器上，而不是使用TSDU更新目标机器，以重新对齐两个机器(D)。接下来，开发人员可以继续使用TSDU进行频繁更新，并且当将来出现系统不匹配时，可以使用Ghost重新为目标机器的硬盘创建镜像。

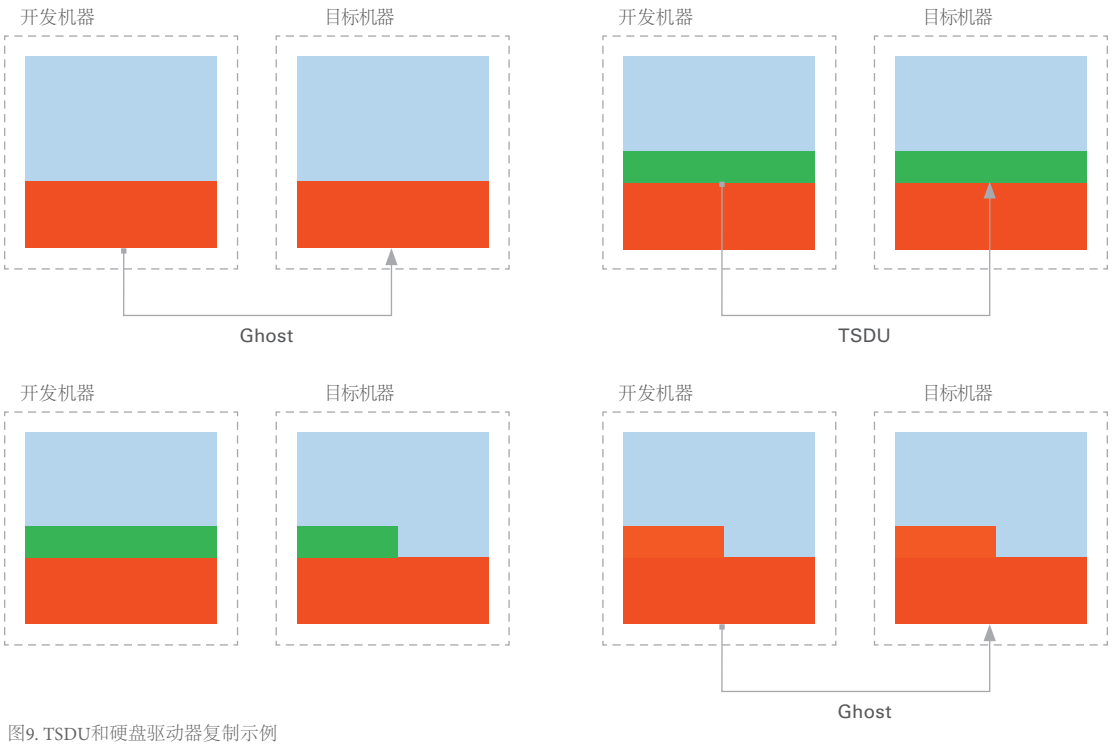

图9. TSDU和硬盘驱动器复制示例

持续集成和持续部署

持续集成(CI）是指通常在单独的CI服务器上持续提交、编译和测试代码。 在大多数测试系统中，CI服务用于为编译、测试和部署系统软件提供必要的框架。 这些服务提供各种配置选项，在CI服务器上定期自动运行，以创建编译计划、自动测试规则和版本部署等。 使用CI服务器最明显的优势之一是跟踪和管理不同编译文件和部署文件的能力。

编译

版本	状态	最后一次编译
1.2	未通过	2016年5月24日
1.1	通过	2016年4月2日
1.0	通过	2015年12月7日

部署

版本	状态	最后一次编译	机器
1.0	通过	2015年6月7日	A
1.1	未通过	2015年6月9日	B
1.1	通过	2016年3月16日	C

表2. CI服务提供用于跟踪应用程序编译和部署的仪表板。

不同的CI工具在功能上的差异很大。开源开发者和软件公司都有开发CI工具。 后者具有为系统设置提供支持的额外好处。

- **Jenkins—**Jenkins誉为“领先的开源自动化服务器”，是当今最受欢迎的CI服务之一，因为它可允许轻松的安装和配置。 Jenkins也可以使用几乎所有的编程语言，因为它可以通过它们的命令行界面或通过广泛的Jenkins插件与程序连接。
- **Bamboo—** 由软件公司Atlassian开发，是领先的专用CI服务。除了Bamboo提供的测试、编译和集成功能外，Atlassian提供了Jenkins所没有的“一流部署支持”。
- **Travis CI and Circle CI—**这两个开源的CI服务提供了极大的扩展能力，但仅与位于GitHub存储库中的项目集成。

总的来说，CI的目标是提供自动化且可配置的工具，使开发人员能够在编译和测试其软件时继续编码。

最佳实践：

基本做法：对于简单系统，组件化通常不是什么大问题。尽管系统使用非常少的代码模块或不使用插件架构，但是每个测试系统通常可以部署为独立应用程序。然而，如果安装时间变得非常久并且开始减慢部署速度，则可能需要采用更高程度的组件化方法，而不需要重新安装所有组件。

高级做法：当测试系统变得庞大、复杂或者使用插件架构时，是时候淘汰单一部署镜像，采用模块化部署来单独更新每个组件。 使用插件架构是实现这种模块化设置的快速方式，但也可以通过配置CI服务来实现。

实际情景

高级部署框架的一个案例是一家音频设备生产公司使用TestStand和LabVIEW对其产品进行功能性电气测试。 音频设备制造商的测试部门有50多个测试系统，遍布在全球各地。 每个系统使用一个PXI机箱来容纳各种模块，包括数据采集、数字I/O、数字信号采集、数字万用表和频率计数器板卡。

负责部署的测试工程师按照列出的步骤对每个要上线的新测试系统进行操作。

1.创建基本系统镜像

每个新测试系统都会有一个必要软件的列表，包括公司自己开发的软件和第三方软件，以确保系统的安全性。该公司的IT部门需要此软件，包括防病毒软件、VPN安全应用程序和Windows组策略配置规范。 其次，每个系统需要一个基本软件集来执行其必要的测试序列。这个软件的主要组件是一组与公布的NI系统驱动程序集进行交叉检查的驱动程序。也就是说，一个版本的测试系统可能包含NI-DMM 14.0、NI-Switch 15.1、NI-FGEN 14.0.1和NI-DAQmx 14.5驱动程序。 此外，还需要LabVIEW 2014和TestStand 2014的运行引擎来运行主测试系统可执行文件。 下面的图表概述了所有必要的软件。

软件	版本
NI-DAQmx驱动程序	14.5.0
NI-DMM驱动程序	14.0.0
NI-Switch驱动程序	15.1
NI-FGEN驱动程序	14.0.0
LabVIEW运行引擎	2014
TestStand 运行引擎	2014
内部防病毒软件	3.2

表3.创建基本系统镜像时，重要的是明确列出所需的驱动程序和运行引擎版本。

部署到新测试系统的第一步是在开发机器上创建此镜像，并使用硬盘驱动器镜像软件复制该镜像。 此软件镜像可能是之前创建的，因而有可能在多个机器上重复使用。 这有助于降低部署成本，因为每批次相同的测试机器只需要进行一次安装。

在通过安装所有必需的软件生成基本系统镜像后，使用Symantec Ghost来复制硬盘驱动器并将新镜像上传到编译服务器。 编译服务器位于总部，并且只需要维护存储位于服务器上的多个系统镜像的大容量存储器。

2.部署基本镜像

将基本镜像上传到编译服务器之后，测试工程师会将新的测试系统连接到公司网络，然后使用web界面连接到镜像服务器并浏览可安装的各种基本系统镜像。在选择适当的版本后，Symantec Ghost使用镜像对新系统的硬盘驱动器进行镜像。 此时，测试系统具有执行测试序列所需的基本必需软件。

3.验证硬件

在将必要的硬件模块物理安装到PXI机箱并打开系统后，测试工程师需要在软件中将系统别名映射到实体设备。 尽管提供模块与关联插槽编号的列表，测试工程师必须使用配置系统设置来映射别名，以便模块位置可以在系统之间移动。 对于该公司，每个测试系统使用工程师编辑的.ini文件，提供实体系统硬件至测试系统别名的映射。 这通过在MAX中识别设备并手动编辑.ini文件以创建相应的映射来实现。

4.安装应用程序和组件

此时，测试系统已经安装了基本系统镜像并验证了实体硬件。现在，工程师的任务是安装最新版本的测试应用程序。在这种情况下，应用程序是由TSDU生成的TestStand安装程序，包含了所有必需的代码模块、序列文件和支持文件。为了解释这个安装程序是如何生成的，我们需要看一下生产公司采用的开发系统。每个开发人员在LabVIEW中创建一个特定的测试步骤或在TestStand中创建测试序列，并将它们提交到一个Apache Subversion源代码控制存储库。此存储库位于运行CI服务（Jenkins）的服务器上。 Jenkins服务运行所提交代码模块的测试，接着TestStand序列分析器使用命令行对序列进行验证，然后使用TSDU命令行界面将必要的测试序列编译到安装程序中。在编译每个安装程序之后，使用Jenkins Deploy Plugin将安装程序及其必需的支持文件自动部署到编译服务器上。

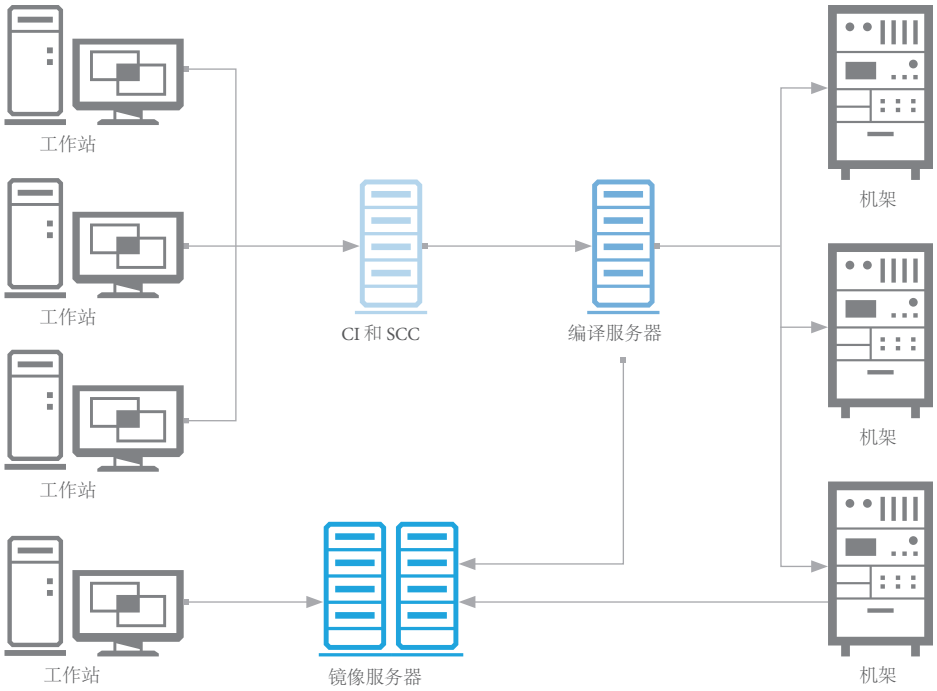

图10.该测试部署系统使用镜像服务器来存储和部署基本系统的镜像，以保持各种测试站彼此同步。然后，开发人员定期将源代码上传到持续集成和源代码控制服务器，以定期编译和测试提交的代码。一旦提交的代码通过所有必要的测试，内置镜像将添加到编译服务器，由编译服务器处理该测试软件系统镜像的大规模发布。

5.执行

将TestStand安装程序放在编译服务器上后，测试工程师可以将安装程序下载到新的测试系统上。 然后，工程师可以运行安装程序，找到主测试可执行文件，并开始运行基本测试系统。

使用这个部署系统，测试工程师可以快速轻松地对每个测试系统进行更改。硬盘驱动器镜像系统可用于大规模代码修订或驱动程序集升级，而较轻量级的编译服务器可用于部署对主测试应用程序或单个组件和插件的小更改。

总结

测试系统部署通常可能是一个复杂的过程，尤其是随着测试系统的复杂性和数量不断增加。在开发测试系统的早期建立正确的部署过程是实现可扩展的成功部署的关键。创建成功部署过程的第一步是要确定和定义所有必需的测试系统组件，并且采用正确的部署方法。动态硬件配置选项也是许多部署系统的重要考虑因素。对于更大型的高级系统，动态解析部署镜像和目标机器之间的依赖关系有助于降低部署过程的复杂性以及升级或重新镜像系统所需的时间。管理和测试部署镜像的每个版本是测试系统开发人员需要考虑的另一个重要因素。无论是采用持续集成服务还是配置文件，都必须维护一个可扩展的版本管理系统来实现分布式部署。测试系统的部署方法必须针对系统的功能和性质进行高度自定义。本指南的各个章节提供了构建可扩展解决方案所需的建议，不管使用何种工具或系统有哪些功能。

TestStand Deployment Utility

The TestStand Deployment Utility可自动化执行部署中的许多步骤，包括收集测试系统的序列文件、代码模块和支持文件，然后为这些文件创建安装程序，从而简化了部署TestStand系统的复杂过程。

了解有关TestStand Deployment Utility的更多信息

LabVIEW Application Builder最佳实践

LabVIEW Application Builder最佳实践可简化LabVIEW应用程序的管理和组织。 这些可建议帮助工程师在开始开发之前制定指南和步骤，以确保其应用程序适用于大量VI和多个开发人员，从而节省开发时间和资源。

开始使用Application Builder最佳实践开发LabVIEW项目