境之后的构件正确性。虽然认证在许多行业成为惯例，但是在传统的软件行业还尚未采用，微软都从没有去认证Windows而为死机承担过责任，那么方兴未艾的构件产业该怎么办？

如何解决适配构件的复杂性

CBSE的需求工程更加复杂，因为构件通常需要满足多个系统的需求，所以构件可配置性是构件的基本特征之一，同样，为了在特定的应用系统中使用而改变构件的过程，即适配活动是使用构件的必要活动。

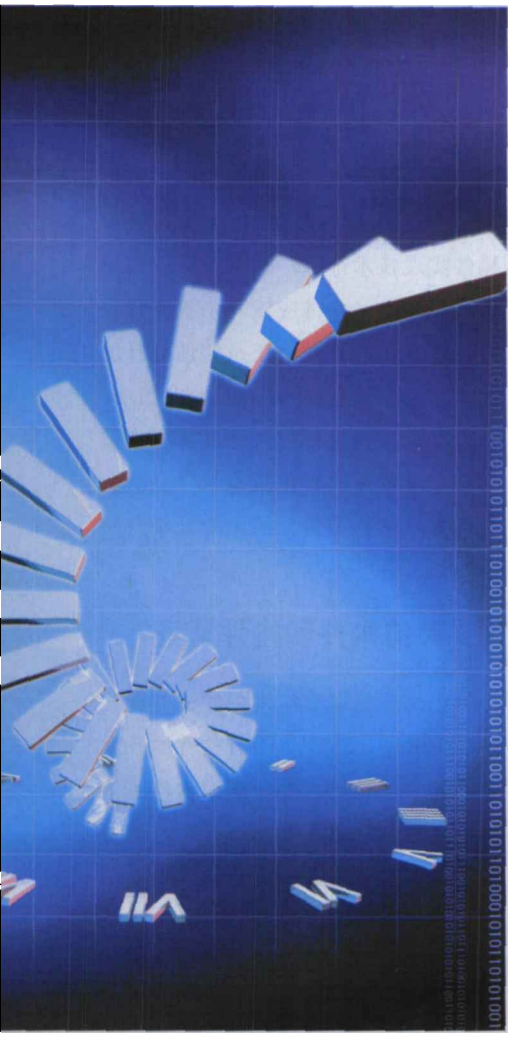
构件的适配源于两个方面原因：构件接口语法的不匹配；构件的功能和行为不完全满足需求。所以，适配的方法也是围绕这两方面展开。常用的适配方法有活动接口方法、二进制构件适配方法、继承方法、就地修改方法和叠印方法，其中的白盒适配方法不应该成为主流。另一种较好的方法使用产品线体系结构来实施构件配置法则。

开发工具准备好了吗？

实用的CASE工具是CBSE成功的根本，像Visual Basic这样开发工具都曾是非常成功的构件开发平台。但是，对CBSE而言，许多种类的工具仍未出现：构件评价工具、构件库管理工具、构件的测试工具、基于构件的设计工具、运行时系统的分析工具、构件的配置工具等等。CBSE的目标是简单且高效地用构件创建系统，这个目标只能在具有广泛的工具支持下才能实现。软

构件可测试性挑战

|上海软件构件化服务中心 编译



构件的可测试性是设计和测试软件程序及构件的重要概念之一。运用具有良好的可测试性的程序和构件来构建软件，可以简化测试操作、减少测试开销、提高软件质量。

James Bach曾指出，有一些程序特性可以用于可测试的软件，包括可操作性、可观察性、可控制性、易理解性等等。Jeffrey M. Voas和Keith W. Miller将软件可测试性看成是可靠性的三个难题之一。他们指出软件可测试性分析对于检测和评估一个使用经验主义(empirical)分析方法的软件测试是很有用的。

然而，在构件工程、基于构件的软件开发中，开发者对构件的可测试性仍然存在一些疑惑。什么是构件的可测试性？它的相关要素是什么？怎样检查、测量或评价软件构件的可测试性？怎样设计和开发可测试的构件来达到良好的可测试性？

构件可测试性是什么？

在构件工程中，有几种不同的构件可测试性的观点，包括构件可观察性(observability)、构件可跟踪性(traceability)、构件可控制性(controllability)和构件易理解性(understandability)。

构件可观察性和可跟踪性

根据Roy S. Freedman的观点，软件可观察性是指根据一个程序的操作行为、输入参数及输出结果，来察看观察这个程序的简单程度。这也暗指了设计和定义一个构件的接口(例如输入、输出接口)将会影响到它的可观察性。可以使用Roy S. Freedman提出的方法来检测一个构件的接口，相对于它的输入，来评价观察它的操作和输出的简单程度。应用到构件工程中，构件可跟踪性是另一个影响构件可观察性的要素。

软件构件的可跟踪性是指跟踪构件属性和构件行为的嵌入能力的范围。它有两方面：行为跟踪(behavior traceability)和跟踪可控制性(trace controllability)。行为跟踪指构件跟踪它的内部和外部行为的便利的程度。现实世界中，构件工程可以通过在软件中增加一个程序跟踪机制来检查和监视软件构件的内部和外部行为。

共计有六种类型的构件跟踪，它们是操作、性能、错误、状态、GUI事件和通讯跟踪。可以使用不同的方法将跟踪可控制性添加到软件构件中。为了支持嵌入功能的访问，必须在构件中定义一个标准跟踪接口。构件跟踪接口的标准化和跟踪格式对一个基于构件的程序建立系统化的跟踪解决方法很重要。

构件可控制性

一个程序（或构件）的可控制性是一个重要的特性，它表明了控制一个程序（或构件）的输入／输出、操作和行为的简单程度。构件开发者从三个方面察看软件构件的“可控制性”：行为控制，特性定制和安装布置。第一个方面与构件的行为和输出结果（相应于操作和输入数据）的能力有关，第二个方面指支持构件内部特性的定制和配置的嵌入能力，最后一个方面是指构件安装和配置的控制能力。

构件可理解性

构件可理解性依赖于构件信息提供的多少以及它们表述的好坏。构件文档的表述是第一要素。

构件可理解性的第二要素是构件程序资源表述，包括构件源代码及其支持元素，例如安装代码和测试驱动等。

最后一个要素是构件质量信息的表述，包括构件验收的测试计划和测试套件，构件测试度量和质量报告。

虽然看上去构件供应商隐藏了详细的测试信息和问题信息也挺合理的，但是用户还是希望在不久的将来他们能够提供构件的质量信息、验收的测试计划，甚至是测试套件。

测试软件构件的挑战

在构件工程范例中，主要目标之一是产生可复用的软件构件作为软件产品。第三方工程师根据用户给出的需求，使用构件作为构建软件系统的部件。所以，程序的可测试性很大程度上依赖于相关构件及其集成的可测试性。构建软件构件的测试要考虑如下几点：

怎样复用构件测试？

考虑软件构件的演化，必须注意构件测试的可复用性。复用构件测试的关键是开发一些系统化方法和工具，来建立可复用的构件测试套件，管理和存储各种测试自愿，包括测试案例、测试数据和测试脚本。对于工程师来说，使用ad-hoc方法，用相同的测试套件技术来处理与当前软件不同的软件构件（例如，第三方构件）很难。在构件验收测试和构件集成中，这个问题就会影响构件测试的复用。

解决这个问题有两个可选择的方法。第一是为构件建立新的plug-in-and-test的测试套件技术；另一个方法是在构件内部建立测试，也就是嵌入式测试。第一种方法是在构件外部的测试套件中建立和维护构件测试，第二个方法是在构件内部建立构件测试。显然，如果一个友好的测试操作接口可行的话，这种方法简化了构件测试，减少了用户方的构件测试开销。要执行嵌入式构件测试，我们需要其他功能工具来执行测试、测试报告和测试结果检查。所以，需要标准化测试访问接口，来支持构件、测试套件和嵌入式测试的交互。

如何构建可测试的构件？

一个理想的可测试软件构件不仅是可配置可执行的，而且在标准化构件测试工具的支持下也是可测试的。与普通构件不同的是，可测试构件有以下一些特性：

可测试构件一定可跟踪；可测试构件一定有一些很好定义的测试工具的嵌入接口；具有嵌入式测试的可测试构件必须使用标准化机制；

关于可测试构件的设计有三个问题。第一个问题是对可测试构件，怎样设计和定义通用架构和测试接口，第二个问题是怎样用系统化方法产生可测试构件，最后一个问题是为了支持测试和可测试构件，怎样控制和最小化程序的费用和资源。

如何构建构件测试驱动和存根？

在设计应用中，工程师基于给定的需求和设计说明书，使用ad-hoc方法来开发特定模块或特定产品的测试驱动和存根。这种方法的主要缺点是产生的测试驱动和存根只对特定的项目（或产品）有用。

很明显，传统方法会导致在构件测试驱动和存根构建上的更高的开销。

显然，需要新的系统的方法来为不同的构建和各种定制服务构建测试驱动和存根。这其中的关键就是为构件产生可复用的、可配置的（或可定制的）、可管理的测试驱动和存根。

软件构件的可测试性也是决定构件质量重要因素之一，为了开发高质量构件，应该研究构件测试和构件的可测试性。

构件测试驱动必须是基于脚本的程序，只使用它的黑盒功能。这有两组，第一组包括特定功能测试驱动，每一个都使用了一个构件的特定产生功能（或操作）；第二组包括特定情形的测试驱动，每一个都使用了一个构件黑盒操作（或功能）的特定顺序。

在构建构件框架时需要构件测试存根。每一个测试存根模拟构件的一个黑盒功能和／或行为。有两种方法产生测试存根。第一种是基于模型的；第二种方法是操作化的基于脚本的。然而，怎样用系统化的方法产生这些特定功能的测试存根是一大挑战。

总体而言，一个程序测试执行环境包括几个支持功能：测试检索、测试执行、测试结果检查和测试报告。显然，一个构件的测试环境必须包括这些类似的功能。软