测试嵌入式系统
—您使用“测试颗粒”了吗？

Vincent Encontre

Rational Software 白皮书

TP 317, 11/01

Rational®
the software development company

目录

简介

本文首先对测试嵌入式系统进行了总体介绍，然后讨论嵌入式系统的问题如何影响测试过程和技术，以及 Rational Test RealTime 如何提供这些问题的解决方案。

一些常用概念的定义

让我们从一些常用概念的定义开始。

什么是测试？测试是训练有素的过程，包括检查应用程序及其组件的运行状态、性能以及健壮性是否达到预期的标准。虽然通常并不明确说明，但其中的一个主要标准是应用程序要尽可能地达到零缺陷。因此，预期的运行状态、性能以及健壮性都应该有正式的描述并且可以进行测量。逐个调试意味着除去缺陷（即“错误”）且被认为是测试过程的唯一部分。

究竟什么是“嵌入式系统”？要给出它的精确定义，是一个很难且颇有争议的问题，以下是一些示例。嵌入式系统是已渗透入日常生活的非常“聪明”的设备，例如您口袋里的移动电话以及它所涉及到的所有无线基础结构；桌面上的 Palm Pilot；电子邮件所通过的因特网路由器；大屏幕的家庭影院系统；空中交通控制站，以及它所监视的误期航班！如今软件在这些设备中所占的价值已达到 90%。

图 1 运行在嵌入式软件上的世界

虽然不是全部，但绝大多数的嵌入式系统都是“实时”的。这两个术语，即“实时”和“嵌入式”通常被互换地使用。 实时系统是那些计算的正确性不仅取决于计算的逻辑正确性，同时也取决于产生结果所花费的时间的系统。如果不能满足系统的时间限制，就会出现系统失败的情况。但对于一些确定为“安全关键”的系统来说，决不允许失败。因此，对于嵌入式系统，***测试时间限制和测试功能运行情况同样重要***。

虽然嵌入式系统的测试过程和很多用于其他类型应用程序的测试过程很相似，但有些对嵌入式世界很重要的问题会影响测试过程：

- 应用程序开发与执行平台的分离
- 种类众多的执行平台和交叉开发环境

- 广泛的部署体系结构
- 各种实施范例共存
- 执行平台上的稀缺资源和时间限制□ 缺乏设计清晰的模型
- 形成质量和认证标准

本书的结尾部分有关于这些问题的更多详细讨论。

这些问题对测试和衡量嵌入式系统的能力有很大的影响。同时也解释了为什么很难测试嵌入式系统，以及由此导致的测试嵌入式系统成了当今的开发实践中最薄弱的环节。所以，最新的研究结果（请参阅[R1]）也就不足为奇了：50% 以上的嵌入式系统开发项目都会落后进度几个月，并且只有 44% 的设计属于能够在功能和性能都满足期望的那 20% — 即使 50% 以上的开发工作都投入在测试上。

在本书的剩余部分，我们将：

- 浏览常规测试迭代，从中获取一些基本的、测试所必需的技术。
- 用具体实例说明此迭代如何测试复杂系统（如在嵌入式世界发现的那些），并在考虑到这些事项的基础上，向理想测试技术添加功能。
- 调查究竟是什么使嵌入式系统难于开发和测试
- 对这些添加到功能列表中的功能是如何使用相应的测试技术来实现的进行评估

Rational Test RealTime 将用作实施大部分此理想技术的一个产品示例。

常规的测试迭代

准备“测试颗粒”

在任何测试中，第一个必做的步骤即是确定要测试的颗粒。使用单词“颗粒”是为了避免使用其他诸如组件、单元或系统等这些缺少概括性定义的词。 例如，在 UML 中，组件是应用程序的一部分，并且具有它自己的控制线程（即一个任务或一个 UNIX 进程）。我也喜欢使用单词“颗粒”作为粒度的基础，因为粒度很好地表示了可以测试的许多元素 — 从一行代码到一个巨大的分布式系统。

确定要测试的颗粒，然后将它转变为可测试的颗粒，或“测试颗粒”（GuT）。此步骤包括将颗粒从它的环境中隔离开来，这需要借助于存根，有时是*适配器*的帮助而使颗粒独立。 存根是一部分代码，这些代码能够模拟颗粒和其他应用程序之间的双向访问。

然后构建一个测试驱动程序。它以适当的输入激发 GuT，然后评估输出的信息，并将该信息与预期的响应进行比较。适配器用于使测试驱动程序激发 GuT。按照 GuT 中指定的阈值进行激发和评估：我们称之为“控制点”和“观察点”（PCO），该术语直接取来自电信行业。PCO 可以位于 GuT 的边界或内部。

C 函数颗粒的 PCO 示例为：

- 颗粒内部的观察点：函数中某个特定行的内容
- 颗粒边界的观察点：函数所返回的参数值
- 颗粒内部的控制点：局部变量的变更
- 颗粒边界的控制点：使用实参的函数调用

存根和测试驱动程序可以是应用程序的其他部分（只要可用）而不一定必须进行开发才能测试 C 函数或 C++ 类。可以从应用程序的其他部分访问或激发 GuT，然后该部分应用程序即作为存根或测试驱动程序。存根和测试驱动程序构成了 GuT 的测试装置环境。

描述测试用例

描述测试用例即为处理以下内容：

- 合适的 PCO — 这取决于要执行的测试类别，例如功能测试、结构测试、负载测试等等
- 和如何利用它们 — 通过它们发送什么信息，或期望通过它们接收到什么信息，如果有，又是以什么顺序

测试的类型，像发送的或预期接收的信息一样，是由 GuT 的需求集所决定的。对于“安全关键”的系统，正式且精确的需求是开发流程的主要部分。虽然正式的需求是测试的重要驱动来源，但有时候它们并不能明确地确定能够在系统中发现重要缺陷的测试。使用正式的需求，对于那些缺少特定需求的不是很重要的应用程序，测试人员必须考虑引进一些适当的测试（也称为“测试构想”），从而推断出一些需求用于测试 GuT。必须将这些需求转变为利用可用的 PCO 的正式测试用例。“正式”的意思，简单地说，即为*可执行*。

通常，需求本身不是正式的，也不会自动转变为正式测试用例。该转变过程经常会引入错误，因此测试用例就不能精确地反映需求。自从引入 UML 后，规范语言更正式了，而且如今也可以用来表达基于需求的正式测试用例，从而避免进行转变而生成的问题。Rational QualityArchitect 和部分 Rational Test RealTime 提供了使用诸如基于模型的测试技术之类的有效示例。

不幸的是，并不是所有需求都是使用 UML 描述的，尤其是在嵌入式世界中：测试用例最常用的正式描述技术只使用编程语言，例如 C 或 C++。因为 C 和 C++ 的广泛普及（因此减少了学习时间），所以就不必考虑测试用例需求，例如PCO 定义或预期的 GuT 运行状态。因此，也不会要求您写详尽的测试用例。特定的高级别的测试语言的设计早已解决了这个问题，这些测试语言非常适合特定的测试领域，例如数据集中测试或基于事务的测试。Rational Test RealTime 提出了本机 3GL 和专用的高级别脚本语言的复合体，它体现了两方面的优点 — 既减少了学习时间，又提高了编写测试用例的效率。

另一个非常有价值而且很有效率的编写测试用例的方法是使用会话记录器。当 GuT 被激发后（无论是手动还是通过它的将来环境），特定的观察点在 GuT 周围记录信息，并且这些信息会进一步自动转变为以后重放的合适的测试用例。在 Rational Rose RealTime 中即可发现该类会话记录器的例子，在Rational Rose RealTime 中，模型执行会生成反映跟踪执行的 UML 时序图，然后将此时序图作为 Rational QualityArchitect RealTime 的测试用例模型。每个测试用例都必须将 GuT 带入允许测试运行的特殊启动状态。测试用例的这一部分称为前同步信号。在有效的测试结束时，不论结果如何，测试用例脚本都必须将 GuT 带入允许执行下一个测试用例的最终静止状态。测试用例的这一部分称为后同步信号。

部署和执行测试

一旦被描述，测试用例即被转变和整合为测试驱动程序和存根的信息（相对于操作）部分。值得注意的是，存根的描述是测试用例的一个完整部分。测试用例在测试装置执行过程中执行。

观察测试结果

测试执行的结果是通过“观察点”来监视的。 在颗粒的边界，“观察点”的典

型类型包括：

- *函数或接收*的消息所返回的参数
- *全局变量的值*

- *信息的排序和定时*

在颗粒的内部，“观察点”的典型类型包括：

- *源代码内容*，此内容提供关于涵盖了 GuT 哪个软件部分的详细信息
- *控制图*以跟随已执行的各种逻辑分支。
- *信息流程*以便使 GuT 的各个部分之间随时间变化的信息交换可视化。 通常，这类流程在 Rational Test RealTime 中被表述为 UML 时序图。
- *资源使用*情况显示非功能性信息，例如在 GuT 的不同部分中花费的时间。内存池管理，或事件处理性能。

所有这些观察结果可以收集起来用于一个测试用例，或者聚集起来用于一系列测试用例。

决定下一步

一旦收集和综合了所有测试数据，将产生两种结果：一个或多个测试用例失败，或所有测试都通过。

测试用例失败的原因有很多：

- *不符合需求*（包括暗含的零崩溃需求） – 您必须返回至实施来解决 GuT 中的问题，甚至从设计开始。
- *测试用例错误* – 这比我们通常想象的更频繁发生。测试，像软件一样，并不总像预期的那样第一次使用就成功。 修改测试用例来解决此问题。
- *测试用例无法执行* – 仍像软件一样，看起来所有地方都没有问题，但就是无法部署、启动或将测试装置连接到 GuT。

如果所有测试都已通过，您可能想要考虑以下这些操作：

- *再次评估您的测试* – 如果您处于测试过程的早期阶段，可能会对该测试的价值和目标产生疑问。 测试应该发现问题，尤其是在开发过程的早期阶段，如果您没有发现任何问题，则…
- *增加测试用例数量* – 这应该能增加 GuT 的可靠性。反对的理由是可靠性本是需求的一部分，这是正确的。但是，可靠性的级别通常直接与测试用例的整体设置的 GuT 覆盖级别有关。
- *代码覆盖是应用范围最广的覆盖类型* – 如在 Rational Test RealTime 中所实施的那样。基于代码覆盖的测试有助于定义其他测试用例，这些用例会增加覆盖级别，直到达到需求的级别。这部分测试通常称为结构测试，测试用例基于 GuT 内容，而不是直接取决于需求。
- *通过聚集颗粒来增加测试范围* – 如下段中所述，您可以将此常规过程应用到系统更大的部分。

测试何时结束？

对于软件从业者来说，这是一个一直存在的问题，也不奢望在本书完全解决它。但是，可以利用的探索性方法是考虑所测试系统的安全的重要性。该系统是否可被视为“安全关键”？ 对于非安全关键系统，可以根据一些主观条件（例如“上市时间”、“预算”以及“足够好”等等）来决定是否可以结束测试。然而对于决不允许失败的“安全关键”系统来说，若根据这些条件来决定结束测试，后果将是无法想象的。在本书近结束处的标题为*“形成质量和认证标准”*的部分，有关于处理此问题的一些建议。

常规测试技术的需求

从先前描述的常规测试迭代中，我们可以推断出测试工具 必须实现一些基本功能。它们必须：

- 帮助定义并隔离 GuT
- 提供测试用例表示法，即 3GL、可视部件或高级别的脚本，并且该脚本能够支持 PCO 的定义、发送至 GuT （或期望来自于 GuT）的信息的定义、以及前同步信号／后同步信号的定义
- 帮助从需求或测试构想中精确地衍生出测试用例 使用会话记录器提供实施测试用例的其他方法
- 支持测试用例部署和执行
- 报告观察
- 访问成功或失败

Rational Test RealTime 支持这些功能，除了这些需求以外，还可以测试嵌入式系统领域中的复杂系统。

测试复杂系统的六个递增步骤

复杂系统的常规体系结构和实施

嵌入式系统是复杂系统，它由种类繁多的体系结构组成，小到微小的 8 位微控制器，大到巨大的由多个处理器平台组成的分布式系统。但是，三分之二的系统运行在实时操作系统（RTOS）上，并实施扩展为 RTOS 任务或进程的线程概念，这些实时操作系统可以从货架上或商店内获得。（线程是具有独立的控制流的颗粒。）在 UML 中，此概念被称为“组件”，而节点是指独立地处理运行 RTOS 所管理的一系列任务的单元。节点之间的任何通信通常都由消息传送协议（例如 TCP/IP）来执行。

嵌入式系统的绝大多数开发人员都使用 C，C++，Ada 或 Java 作为编程语言（2002 年 70% 的开发人员 使用 C，60% 使用 C++，20% 使用 Java，5% 使用 Ada，在 [R1] 中有注释）。在同一个嵌入式系统中发现多种语言是很常见的，尤其是 C 和 C++ 同时使用，或 C 和 Java 同时使用。通常认为 C 是最有效和最接近平台的详细描述语言，而 Java 或 C++ 则更多产，这要感谢面向对象这一概念。但是，应该注意的是，嵌入式系统程序员并不是对象的崇拜者。

在嵌入式系统环境中，颗粒可以是以下某个对象：此列表按复杂性递增排列。

- C 函数或 Ada 过程
- C++ 或 Java 类
- C 或 Ada（系列）模块
- C++ 或 Java 类集群
- RTOS 任务
- 节点
- 完整的系统

对于最小的嵌入式系统而言，完整的系统仅由一系列 C 模块组成，并不集成任何 RTOS 相关的代码。对于最大的系统（分布式系统），网络协议增加了复杂性。 下一部分显示了此常规体系结构是如何影响各个测试步骤的。

测试的六个递增步骤

由颗粒的不同类型所决定，并根据本部分稍后讨论的行业中的常规使用情况，以下这六个测试步骤对于检查应用程序的运行状态、性能以及健壮性是否达到预期标准是必需的。这些步骤为：

- 软件单元测试
- 软件集成测试
- 软件验证测试
- 系统单元测试
- 系统集成测试
- 系统验证测试

软件单元测试

GuT 要么是隔离的 C 函数，要么是 C++ 类。根据 GuT 的用途，测试用例由以下任意一个组成：

- *数据集中测试* — 对函数参数值应用大量不同的数据，或
- *基于场景测试*— 执行不同的 C++ 方法调用序列以执行需求中发现的所有可能用例

观察点返回值参数、对象属性评估以及源代码内容。白盒子测试用于测试单元，这意味着测试员必须熟悉 GuT 的内容。开发人员负责单元测试。

这是因为在复杂的嵌入式系统中，很难发现微小的错误，必须在单元测试级别不遗余力地找到并除去它们。

软件集成测试

GuT 现在是一系列函数或一群类。集成测试的本质是验证接口。 同一类型的控制点适用于单元测试（数据集中主函数调用或方法调用序列），而观察点则主要适用于使用信息流程图的低级别颗粒之间的交互。 一旦 GuT 变得有意义，即“端到端”测试场景可以应用到 GuT 时 — 就可以运行性能测试，它可以很好地说明体系结构的有效性。对于功能测试，则是越快越好。每个即将到来的步骤都包括性能测试。白盒子测试也是此步骤中要使用的方法。开发人员负责软件集成测试。

软件验证测试

GuT 是组件内部的所有用户代码。这可以视为软件集成的最后一步。用例现在正接近最终用户的场景，同时远离了实施的细节。由于 GuT 是整个系统的重要部分，观察点包含资源使用情况评估。此时，我们仍将该步骤视为白盒子测试，也是最后一个。软件验证测试仍是开发人员的职责。

系统单元测试

GuT 现在是一个完整的系统组件；即在软件验证测试过程中测试的用户代码加上所有 RTOS — 以及与平台相关的部分：任务机制、通信、中断等等。控制点协议已不再是函数调用或方法调用，而是使用例如 RTOS 消息队列接收或发送的消息。

消息传送范例中发现的对称性表明测试驱动程序和存根之间的差别在此阶段可以认为是不相关的。我们将它们叫做虚拟测试员，这是因为它们每个都能替代和充当 GuT 相关的其他系统组件。“模拟器”和“测试员”是“虚拟测试员”的同义词。虚拟测试员技术具有通用性，应该能够适应众多的 RTOS 和网络协议。从现在开始，测试脚本通常将GuT 带入所需的初始状态，然后生成消息样本的有序序列，并验证接收到的消息，

方法是将消息内容和接收日期与预期的消息内容和限制时间作比较。在各个虚拟测试员处分发和部署测试脚本。监视系统资源以便评估系统维持嵌入式系统执行的能力。从该步开始，灰盒子测试是首选的测试方法。所需的全部知识即是与 GuT 的接口知识。系统单元的测试取决于组织，可以是开发人员的职责，也可以归系统集成团队负责。

系统集成测试

GuT 从单一节点内的一系列组件开始，最后包含所有系统节点，甚至到一系列分布式节点。 PCO 是 RTOS 和网络相关的通信协议的结合体，例如 RTOS 事件和网络消息。除了组件以外，虚拟测试员也可以充当节点。因为对于软件集成，关键是验证各种接口。灰盒子测试是首选的测试方法。系统集成测试是系统集成团队的职责。

系统验证测试

GuT 是最终全部完成的嵌入式系统。最终步骤的目的有几个：

- 满足最终用户的功能需求。注意最终用户可能是电信网络中的一个设备（比如我们的嵌入式系统是因特网路由器），或一个人（如果系统是消费者所用的设备），也可能两者兼有（可由最终用户管理的因特网路由器）。
- 执行最终的非功能性测试，例如负载和健壮性测试。可以复制虚拟测试员来模仿负载，或进行编程以在系统中产生故障。
- 确保与其他已连接设备的互操作性。检查可应用的互连标准的一致性

详细讨论这些问题已超出本书的范围。黑盒子测试是首选的方法 — 测试员专注于既频繁又危险的用例。

决定如何排列这些步骤的顺序

如今我们有 6 个递增的步骤，如何使用它们？有很多可以用于决定的标准：

- 是否所有这些步骤都适用于您的系统
- 是否应该对所有系统执行选定的步骤，还是只对部分系统执行 选定的步骤的顺序应该适用于选定的部分

综合这些标准，决定主要取决于您开发的嵌入式系统的种类：是否是安全关键的、上市时间、少量还是众多部署等等。以后会写一本专门关于帮助您完成选择过程的指南。本书完成后，您可以通过 Rational Developer Network 进行访问。

处理这些测试步骤的另一方法是将它们合并为一个！验证测试可以视为测试一个更大的 GuT 的单元。集成也可以在验证测试过程中检查。其实问题是，您如何访问 GuT、可以插入何种 PCO 和在哪里插入，以及如何从测试用例确定 GuT 的特定部分（可能很远）。但我们先不管它们，而开始讨论其他问题…

复杂系统测试技术的其他需求

要应对测试复杂嵌入式系统的挑战，测试技术必须添加以下功能：

- *管理多种类型的控制点* — 要激发 GuT，可以使用函数调用，方法调用和消息传递，或通过不同语言绑定（例如 Ada、C、C++ 或 Java）的远程过程调用（RPC）
- *提供多种观察点，*例如参数和全局变量检查；声明、信息、控制路径跟踪；以及代码内容记录或资源用途监视。所有这些观察点都应该提供预期的和实际的能力评估。

嵌入式系统问题是如何影响测试过程和技术的

在本部分中，我们将重点讲述嵌入式系统的特定问题，以及当使用 Rational Test RealTime 作为测试工具时，对这些问题是如何影响用于测试它们的技术进行评估。

应用程序开发与执行平台的分离

在嵌入式系统的众多定义中，其中有一个是这样表述的：

> *嵌入式系统可以是任何软件系统，但它所在的平台与必须与此应用程序本应位于的平台不同，即此应用程序的目标和部署对象是另一个平台。*

从开发角度来说，平台的意思通常就是操作系统，例如 Windows、Solaris 或 HP-UX。应该注意的是，嵌入式系统领域中的 UNIX 和 Linux 的用户比例远远高于（[R1]中注明的是 40%）其他许多 IT 系统领域。目标平台包括之前提到的任何设备。

为什么会有约束？由于可以为最终用户（可以是真正的人或其他设备）对目标平台进行专门的优化和修改，开发就不必针对一些必需的组件（例如键盘、联网以及磁盘等等）。

要应对此双重平台问题，测试工具必须从开发平台以最透明且有效的方法提供对执行平台的访问。实际上，用户必须感觉不到这类访问的复杂性。访问包括测试用例信息的下载、测试执行远程监视（启动、同步，停止）、测试结果以及观察结果的上载。在 Rational Test RealTime 中，所有目标平台的访问都是由 “目标部署”技术控制的。

此外，Rational Test RealTime 在 Windows、Solaris、HP、Linux 以及在设备、嵌入式系统和基础结构产业领头的公司正使用的开发平台上都可用。

种类众多且持续成长的执行平台和交叉开发环境

执行平台有很广阔的范围，从基于微控制器的 8 位电路板，到巨大的分布和联网系统。 由于芯片和系统供应商种类众多，且数量巨大，因此所有平台都需要不同的工具才能开发应用程序。在同一个嵌入式系统中使用多个平台已经越来越普遍了。

通常，为此类环境所做的开发被称为“跨开发环境”。执行平台的多样性即意味着有大量的相应开发工具可用，例如编译器、连接器、装入器以及调试器。

这导致的结果，首先是“观察点”技术只能基于源代码。与对少数本机编译器可用的 Rational PurifyPlus 系列所使用的“对象代码插入”技术相反，Rational Test RealTime 使用源代码检测来处理数字因子。该技术已发展了十年，已产生了高效率的代码检测。

该多样性所导致的另一个直接后果即是跨开发工具的供应商更乐于提供“集成开发环境”（IDE），以此来隐藏复杂性，并使开发人员感觉更舒适。同时，非常需要所有其他工具都能紧密地集成入相应的集成开发环境。例如，Rational Test RealTime 就很好地集成入了 WindRiver 的 Tornado 或 GreenHill 的多集成开发环境。

另一个显著特征是，在日新月异的计算机芯片产业的驱动下，新的执行平台和关联的开发工具的发布十分频繁。这就要求测试技术必须高度灵活，才能在短时间内适应这些新的体系结构。 Rational Test RealTime 无需一周的时间即可完成对新目标平台的目标部署，很多时候两天即可。

执行平台上的稀缺资源和时间限制

按照定义，相对于本应运行应用程序的系统，嵌入式系统只有有限的资源。尤其是在那些可用的 RAM 小于 1 KB 的微小平台上；要建立与开发环境之间的连接，只能使用 JTAG 探测器、仿真器 或 串行链接；或者在微处理器的速度能够处理作业的情况下，使用微处理器。测试工具面临一个两难的选择 — 要么将测试数据放在开发平台上，通过已连接链接发送测试数据，代价是性能低下（通常是无法忍受）；要么由目标平台上的测试驱动程序来解释测试数据。

Rational Test RealTime 使用的技术是将测试装置嵌入目标系统。方法是使用交叉编译器在测试装置内部编译之前已转变为应用程序编程语言（C、C++ 或 Ada）的测试数据，

然后将此测试装置对象文件链接到应用程序的其余部分。使用 Rational Test RealTime makefiles 中的命令行接口会使这个构建链对用户透明。优化代码生成、精简链接映射分配、减少所有工作的内存占用，从而使测试装置在目标平台上所需的资源最小化， 同时提供了以下好处：

- 提高了定时精确性，而且使用了“目标中位置”减少了实时对性能的影响。
- 在少量测试用例执行过程中，避免了在已连接链接上流通任何数据信息主机目标通信。如果需要，观察点数据存储在 RAM 中，并在测试用例真正结束时，通过调试器或仿真器上载。

虽然跨开发环境越来越友善，但目前提供的大部分嵌入式系统的开发和测试仍然十分棘手，这是让很多人头疼的问题。Rational Test RealTime 的目标是简化嵌入式系统开发人员的棘手的开发工作。

缺乏设计清晰的可视模型

嵌入式开发人员喜欢代码！不幸的是，大专毕业生在可视建模方面缺少训练，而且他们倾向于认为代码才是“真正的武器”。尽管可视建模通过使用语言（例如 UML）在将嵌入式知识融入设计方面已取得了很大进步，但大多数嵌入式系统程序员仍喜欢使用普通的原有编程语言工作。而 Java 并没有带来更多的人进行设计！

为了帮助开发人员以他们偏爱的方式工作，Rational Test RealTime 通过提供向导（例如测试模板生成器和 API wrapper）帮助他们根据应用程序的源代码来设计测试用例。此基于代码的测试构建过程的主要优点是能够确保测试适用于应用程序。缺点是无法保证测试用例能够反映它应该体现的需求。很显然，可视建模的广泛采用将减少需求和测试用例之间的差异。 Rational Test RealTime 也为此技术铺平了道路：即为测试用例编译器提供了 Rational Rose RealTime UML 时序图。

形成质量和认证标准

对于某个类别的嵌入式系统 —“安全关键”系统 — 决不允许失败。我们可以在原子核、医疗以及航空等行业发现这些系统。很久以前，为了达到零缺陷的目的，航空行业和政府机构（例如美国的联邦航空管理局）联合起来撰写软件的考虑事项，即 *飞行系统和装备认证*被称为 RTCA 的 DO-178B 标准，[R2] 中有描述。这是世界范围的航空产业在“安全关键”软件开发方面的普遍标准。作为最严格的软件开发标准之一，它也被其他制造“生命第一”的系统的行业部分采用，例如自动化、医疗（FDA 刚刚发行了类似于 DO-178B 的标准）、防卫或交通行业。

DO-178B 根据可能导致或引起系统功能失败的异常的软件运行状态，将软件分为五个关键级别。最关键的是级别 A 的设备，它的故障会为整个系统产生灾难性的失败条件。 DO-178B 包含非常精确的步骤，以便确保级别 A 的设备足够安全，尤其是在测试区域中。 Rational Test RealTime 能满足 DO-178B 测试设备的所有级别的全部强制要求，也包括级别 A 的设备。

摘要

在本书中，我们已简要地介绍了用于测试嵌入式系统的 6 个递增步骤的过程。 考虑到此过程（考虑从很小的系统到很大的系统的所有范围）和具体的特征，以及嵌入式系统的限制，我们推断出一些理想技术要处理嵌入式系统所必须掌握的需求。 Rational Test RealTime 是 Rational 在嵌入式系统领域的新产品，基本实现了这一理想技术。

本书概括地介绍了测试嵌入式系统，今年还将重点针对讨论过的各个主题提供后续文章。

术语

本部分由 Paul Szymkowiak 撰写。

遗憾的是，软件工程世界中所使用的很多术语都有不同的定义。本书中所用的术语可能对在实时嵌入式系统领域工作的人最为常见。但也有一些在软件测试中使用的意义相同或相关的其他术语。在下面的表中提供了对应的术语。

本书中使用的术语	同义的／相关的术语
测试颗粒（GuT）： 为了测试而从自身环境中隔离的元素。	相关术语：测试项、目标、测试目标
控制和观察点： 测试中一个特殊的点，在该点记录测试环境的观察结果、或根据测试的控制流程作出决定。	最相关的术语：**验证点**
后同步信号： 使系统进入特殊的稳定终止状态而采取的操作，是执行下个测试所必需的。	后置条件，复位
前同步信号： 使系统进入特殊的稳定启动状态而采取的操作，是执行测试所必需的。	前置条件，启动
测试装置： 排列好的一个或多个测试驱动程序、特定于测试的存根、为执行一系列相关测试而组合起来的检测装置。	*相关术语：*测试套件或测试驱动程序，测试存根
虚拟测试员： (1) 测试过程中与颗粒交互的 GUT 之外的部分(2) 运行测试的实例，通常代表某人的操作。	*相关术语：*测试脚本或测试驱动程序，测试存根，模拟器，测试员

参考资料

[R1] *Critical Issues Confronting Embedded Solutions Vendors*，*Electronics Market Forecast*，http://www.electronic-forecast.com/，2001 年 4 月。

[R2] DO-178B，*Software Considerations in Airborne Systems and Equipment Certification*，Radio Technical Commission for Aeronautics — RTCA，http://www.rtca.org/，1992 年 1 月。

关于作者

Vincent Encontre 是法国图卢兹的 Rational 新工程中心嵌入式和实时自动化测试业务部的主任。除了旅行、参加会议和回答无数电子邮件，Vincent 潜心研究 Rational Test RealTime，它既是 Rational 产品，又是下一代 Rational 产品的可复用技术。Vincent 在嵌入式建模、测试技术和最佳实践方面具有丰富的经验。在加入 Rational 和 ATTOL 之前，Vincent 先后在 Philips 和 Verilog 公司从事了 13 年的软件工程工具的设计、销售和支持工作，他坚信这些工具有助于更快地构建更好的软件。

在业余时间，Vincent 常与他的三个儿子以及其他人一起踢足球，并喜欢法国南部出产的历史悠久的精美艺术品。

Rational
the software development company

两家总部：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
电话：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
电话：(781) 676-2400

免费电话：(800) 728-1212
电子邮件：info@rational.com
Web：www.rational.com
全球网址：www.rational.com/worldwide