白皮书 | 2016 年 4 月

全自动 ETL 测试：分步指南

ca technologies

第 1 部分

现代组织中 ETL 的重要作用

自其在数据仓库和业务情报领域喷发以来，提取、转换、加载 (ETL) 已成为软件世界中无所不在的过程。顾名思义，ETL 例程包含三个不同的步骤，这些步骤通常并行发生：将数据从一个或多个数据源提取出来；将数据转换成所需状态；将数据加载到所需目标，通常为数据仓库、集市或数据库。经开发的 ETL 例程通常还包括错误处理、记录基础架构和例程环境。[1]

目前，ETL 主要用于准备通常较大量的分散的分析数据和业务情报数据。然而，其用途正在扩大，已不再局限于简单地移动数据，而是越来越多地用于迁移新系统的数据，以及处理数据集成、分类和连接。[2]

在快速发展的现代开发生命周期中，不同发布和多个版本的开发工作在任何给定时间并行开展，ETL 因此成为其中的重要功能。组织必须能够对他们的软件不断增强、集成和创新，为测试人员和开发人员提供正确的数据以便顺利完成每次迭代和发布。数据将从相同的来源提取，但必须经过转换以满足各团队的特定需求。对力求实现“敏捷”或成功实施持续交付的组织，尤其如此。

与 CA Technologies 合作的某大型跨国银行很好地展示了 ETL 在持续交付中的重要作用。该银行当时正在开展收购工作，必须将被收购银行的客户、产品和财务数据迁移到现有基础架构中。这意味着，必须先检索、转换和验证 80 个插入文件，然后才能将文件上传到银行的后端系统。这些数据稍后还必须在 47 个项目之间并行可用，并需保持数据的引用完整性。在此实例中，ETL 在实现成功持续交付所需的并行性方面起到了至关重要的作用。

然而，尽管 ETL 的使用和重要性都在增加，但 ETL 测试反映了测试的总体状况，即该测试过于缓慢且过于手动化，而且还会导致生产中出现大量缺陷。本白皮书将提出在通常采用的典型 ETL 测试方法中遇到的挑战，并进一步探索遇到的其他挑战。然后将提出广泛基于模型的替代方法，思考其如何令 ETL 测试更高效、更有效且更系统化。

第 2 部分

典型的 ETL 测试方法以及遇到的共同挑战

在验证 ETL 转换规则时，测试人员通常会创建影子代码集，并使用该它来转换数据，然后将实际结果与预期结果进行比较。通常，ETL 脚本或 SQL 经手动复制到源数据并运行，并记录下结果。然后将同一脚本复制到目标数据，并记录下结果。然后比较两组结果（实际结果和预期结果），以验证数据是否已正确转换。

根本问题：复杂性和可测试性

这种手动验证方法的根本问题在于，ETL 由于其性质，会快速变得高度复杂。随着企业规模的扩大，其收集的数据的种类和量增加，ETL 规则也需要扩展才能加以应对。在所谓的“信息时代”，这种增长的速度比传统测试方法能够应对的速度要快。事实上，由数据驱动的组织收集的海量信息增速极快，过去两年仅收集了全球 90% 的数据[3]，而普通组织收集的数据量每年都要翻一番。[4]

在设计用于收集、传输、运行和显示这些数据的系统中，每添加一个决策都会使系统的复杂性呈指数增长。这包括 ETL 规则，有很多因素可影响转换的复杂性：

- 所涉及的数据源的数量和种类，包括相关和不相关的数据库类型，以及平面文件；
- 数据目标的数量和种类；
- 从简单的外观到活动联合及规范化等简单和复杂转换的数量和种类；
- 可重用转换、代码段和连接的数量；
- 已创建表格的数量。[5]

当前对近实时解决方案的关注以及这些解决方案带来的额外复杂性，加剧了所有这些因素的影响力。[6]

文档毫无帮助

这一不断增加的复杂性可直接影响 ETL 例程的可测试性。对于 ETL 测试尤其问题重重，因为转换规则通常存储在缺少明确预期结果的拙劣文档中。规则通常设计于开发阶段，而且往往存储在书面文档或电子表格中 — 或者更糟，它们可能仅存在于开发人员和测试人员的大脑中。[7]在这种情况下，根本不存在可从中放心生成测试案例（即影子代码）的真实文档。

在与我们合作的一个业务情报团队中，需求存储为书面文档，而测试案例存储在电子表格中。这种静态文档呈现了一道“文字墙”，必须根据它来破解 ETL 例程的逻辑步骤。文档强调“愉快路径”，不包含任何消极条件，以致于平均系统中需要测试的约 80% 的可能逻辑都被忽略了。这种不完整的存在歧义的文档意味着测试人员无法轻松准确地了解 ETL 例程。

测试人员经常需要填补空白，但当他们误解时，缺陷已经进入 ETL 例程。然后无效数据被复制到目标，即使代码和测试案例就需求文档反映出了似乎合理的解释。

“垃圾进，垃圾出” — 手动推导测试案例和预期结果

事实上，进入生产阶段的 56% 的缺陷都可在需求文档中追溯到歧义。[8]部分原因是测试案例和预期结果均从拙劣文档中手动推导出来：这是一个通常会导致较差测试覆盖率的极其耗时的过程。

质量

手动推导是即兴且非系统化的，因此创建的测试案例通常都是测试人员脑海中随意构想的案例。鉴于已讨论的 ETL 例程的复杂性，以及提供的拙劣文档，期望最具天赋的测试人员创建验证可能的数据组合所需的每个测试，是不公平的。例如，如果某个包含 32 个节点和 62 条边的简单系统采用线性设计，则其逻辑中可能有 1,073,741,824 个例程 — 此数字超出了任何人的想象。

因此即兴推导会导致大量的测试不足和测试过度问题，在此过程中只有 ETL 例程中涉及的小部分可能逻辑经过测试。负面测试将是个特别的挑战，而文档等测试案例通常仅注重愉快路径。但是，我们必须根据这些离群值和意外结果进行测试，因为 ETL 例程必须拒绝这种“不良数据”。

例如，与 CA Technologies 合作的一家金融服务公司采用了 11 个测试案例，而测试覆盖率仅为 16%。此数字十分符合标准，通过我们的审计发现，绝大多数都只有 10-20% 的功能测试覆盖率。而该公司的另一个项目经历了高达 18 倍的过度测试；他们将测试案例堆积起来，期望充分测试系统，但仍然未能达到最大覆盖率。150 个额外的测试案例花了他们 26,000 美元来请外包提供商执行测试。[9]

覆盖率极低的结果是，缺陷进入代码中，而在代码中修复缺陷既昂贵又费时：研究发现，相比尽早发现缺陷并予以解决，修复测试期间的错误需多花 40-1000 倍的资源[10]和多 50 倍的时间[11]。更糟的是，错误可能逃过检测，以致于无效数据复制到实时目标，从而可能威胁系统的完整性。而且，面临静态文档时，测试人员没有可靠的方式来测量测试案例的覆盖率 — 他们根本无法自信地说出给定例程有多少部分在进行测试，也无法根据关键性划分测试的优先级。

时间和精力：测试根本无法满足需求

根据这种文档编写测试案例还是一项耗时费力的工作。在前一个示例中，创建 11 个测试案例需花费 6 小时，而该公司泛滥的过度测试更是耗费了更多时间。在手动测试案例设计上浪费时间之后，在比较实际结果和预期结果时还必须花费更多时间。

鉴于复杂 ETL 例程产生的海量数据，而且源数据通常以各种文件类型存储在各种数据库中，要将大量独立字段与预期结果进行比较会十分耗时。由于转换的数据必须在多个层级进行验证，这更是增加了进行比较的难度。

- 测试人员必须检查数据是否完整，确保数据源计数与目标相符；
- 必须确保数据完整性，并检查目标数据是否与源数据一致；
- 转换必须与业务规则保持一致；
- 必须保证数据一致性，并识别任何意外复制；
- 保持引用完整性，发现任何孤立记录或缺失外键。[12]

有时会做出让步，只验证一个简单的数据集。但是，这也会使 ETL 测试的全面性让步，因此影响到转换的可靠性。鉴于很多 ETL 例程在业务关键型运营中的重要作用，这种让步是不可接受的。手动对比易于出错，这还会进一步影响质量，尤其是如果预期结果未明确定义，或者更糟的是，根本未独立于测试中使用的影子代码进行定义。在本实例中，测试人员倾向于假设测试已通过，除非实际结果特别奇怪：若没有预定义的预期结果，他们可能假定实际结果就是预期结果[13]，因此无法自信地确定数据有效性。

数据问题

到目前为止，我们已经重点介绍推导测试以验证 ETL 规则时遇到的问题。但是，推导出测试案例之后，测试人员还需要贯穿系统的虚拟源数据。这是导致瓶颈和缺陷的另一个常见原因。

您是否拥有测试复杂 ETL 例程所需的全部数据？

拥有足够的“不良”数据对有效进行 ETL 测试至关重要，因为最重要的是，ETL 规则在运行时会拒绝这些数据并以恰当的格式将它们发送给相应的用户。如果未被拒绝，这些不良数据可能导致缺陷或者甚至系统崩溃。

此情况中的“不良数据”可通过很多方式进行定义，这些方式与测试人员必须验证数据时采用的方式相对应。这些可能是基于业务规则不应被接受的数据，例如，在线购物车中的负值（如果未显示代金券）。这些可能是威胁数据仓库引用完整性的数据，例如缺失的相互依赖的或强制性的数据，或者输入数据中缺失的数据。[14]因此，贯穿 ETL 验证规则的测试数据必须包含完整范围的无效数据，才能实现 100% 的功能测试覆盖率。

在仍然向很多组织的测试团队提供的生产数据源中很少发现这种数据。这是因为生产数据是从发生在过去的“一切如常”的场景中提取所得，因此根据其性质已经排除了不良数据。其中不包含意外结果、离群值或进行 ETL 测试所需的边界条件，而是以“愉快路径”为主。事实上，通过我们对生产数据的审计发现绝大多数都只有 10-20% 的覆盖率。讽刺的是，构建越好的例程，允许通过的“不良数据”越少，这意味着用于充分测试 ETL 规则的具有足够种类的数据会更少。

当您需要时，数据是否可用？

进行 ETL 验证的另一个主要问题是数据的可用性。源数据可能提取自企业中的 50 个不同来源。通常 ETL 测试的问题在于，测试被视作一系列线性阶段，在其他团队正在使用数据时，测试团队只有等待。

以银行迁移链为例，其中使用调和工具从一个银行提取数据，然后将其转换到另一个银行的系统。在每个阶段都需要验证数据，检查数据是否已正确转换到财务控制框架，是否已检索账号，还需要进行数小时的纠正工作等等。此过程可能分成多个不同的阶段，从基本输入到重复数据删除，再到数据传播和记录。还可能需要多个团队参与进来，包括 ETL 团队和非 ETL 团队，尤其是处理大型机的团队。

如果企业中的每个数据源的数据无法并行提供给每个团队，则延迟会随着团队闲坐等待而增加。测试人员将编写幽灵代码，然后却没有验证 ETL 规则所需的源数据，因为其他团队正在使用这些源数据。事实上，我们发现测试人员平均需花 50% 的时间来等待、查找、操作或创建数据。这可能占总软件开发生命周期 20% 的比例。

规则改变会产生什么影响？

从静态需求中手动推导测试案例和数据对改变的响应性极差。ETL 例程的变化与业务的发展一样快，业务所收集数据的量和种类也会随之增加。但是，当不断发生这种变化时，ETL 测试却无法满足需求。

可以说，本实例中项目延迟的最大问题是，必须在例程改变时检查和更新现有测试案例。测试人员无法自动确定静态需求和测试案例发生改变所产生的影响。相反，他们必须手动检查每个测试案例，而且无法衡量实际保持的覆盖率。

如业务情报团队在前文提及的，需求和测试案例分别存储在书面文档和电子表格中，因此发生改变尤其会带来重重问题。更改单个 ETL 规则后，一名测试人员需花 7.5 小时来检查和更新一系列测试案例。在与我们合作的另一个组织中，更改需求后，两名测试人员花了两天时间来检查每个现有的测试案例。

第 3 部分

可行的替代方案：全自动 ETL 测试

显然，只要采用手动方式推导测试案以及比较结果，ETL 测试就无法跟上业务需求不断变化的快速步伐。下面介绍一种可能提升 ETL 测试效率和有效性的策略。它是一种基于模型和需求的策略，旨在左移测试工作，从一开始便将质量需求构建到 ETL 生命周期中。这种基于模型的方法将自动化带入每个测试和开发阶段，并使 ETL 测试充分响应不断的变化。

1) 首先从形式模型开始

在 ETL 测试中引入形式建模可体现基本优势，即左移测试工作，将 ETL 规则映射到模型，从该初步工作中即可获得所有后续测试/开发资产。因此，形式模型成为全自动 ETL 验证的基石。

不过，形式建模还有助于解决上文提出的具体问题，包括需求的歧义性和不完整性。尽管 ETL 规则越来越复杂，形式建模可帮助维持可测试性，以便测试人员可直观准确地快速了解需要测试的逻辑。他们可轻松了解应输入哪些有效和无效的数据，以充分测试转换规则，以及各实例中应包含什么预期结果。

例如，流程图模型可将难处理的“文字墙”文档细分为可消化的块。该模型将 ETL 归结为原因和影响逻辑，将其映射为一系列与流程层次结构关联的假设性语句。[15]实际上，每个步骤成为了测试组件，并可与测试人员准确沟通需要验证的内容。因此，将 ETL 例程构建为流程图模型可消除需求文档中的歧义，并避免了其中产生的 56% 的缺陷。

随着 ETL 例程更加复杂，流程图可作为单一参照点。与“静态”书面文档和图表相反，可向模型轻松添加其他逻辑。而且，还可使用子流程技术提取高度复杂的例程，从而提高可测试性。较低级别的组件可嵌入主流程中，以便构成一系列复杂 ETL 规则的大量例程能够整合为单一可视图。

除了减少歧义以外，流程图建模还有助于解决不完整性问题。该模型使建模人员必须考虑约束、负面条件、限制和边界条件，并提出“如果此原因或触发因素不存在会如何？”的问题；因此他们必须系统性地构想消极路径，并调整应在 ETL 验证中占绝大部分比例的负面测试。由于形势模型在数学角度上是 ETL 规则的精确图解，因此还可进一步应用完整性检查算法。

这可消除“悬置 else 问题”(dangling else) 等缺失的逻辑，这样便可推导出覆盖 100% 的可能数据组合的测试案例（但是应注意，存在的组合永远比可作为测试执行的组合要多，因此后面我们将讨论相关的优化技术）。另一个主要优势是，可在模型中定义独立于测试案例的预期结果。换言之，用户可利用流程图定义模型，以纳入边界输入并将预期结果传送到模型中的不同端点。这可以清楚定义验证规则应接受和拒绝的对象，在预期结果不明确的情况下，测试人员便不会错误地认为测试已通过。

应注意的是，采用基于模型的测试进行 ETL 验证并不需要在整个企业中全面采用由需求驱动的测试和开发方法。不需要突然进行彻底改变，而且根据我们的经验，只需 90 分钟即可将 ETL 例程构建为“活动的”流程图模型。然后 ETL 或测试团队可使用此模型进行测试以及重新测试该例程本身。

2) 从流程图模型自动推导测试案例

引入基于模型的测试可自动化 ETL 测试的一个主要手动元素：测试案例设计。测试人员无需再编写幽灵代码，或手动将 SQL 从源数据库复制到目标数据库。相反，流程图中的路径成为测试案例，这些案例可用于将数据贯穿于转换规则中。这些测试案例可系统性地推导出来，而根据静态需求编写代码无法实现此目的。

这种自动推导之所以可行是因为该流程图可覆盖以系统中涉及的所有功能逻辑。然后可运用自动数学算法来确定模型中每条可能的路径，并生成涵盖每个输入和输出组合的测试案例（可通过原因和影响或同伦分析实现此目标）。

测试案例直接关联到模型本身后，便会涵盖其中定义的所有逻辑。因此这些测试案例可提供 100% 的功能覆盖率，于是通过流程图模型获得完整的文档相当于全面测试 ETL 例程。此方法还有一个优点就是，测试变得可测量了。由于可推导出每个可能的测试案例，因此测试人员可准确确定给定的测试案例集合的功能覆盖率是多少。

优化：通过更少的测试测试更多内容

接下来便可运用自动优化算法，以将测试案例的数量减到最少，同时保持最大的功能覆盖率。这些组合技术凭借流程图的逻辑结构才得以实现，其中，原因和影响逻辑（流程图/测试组件中的块）中的一个步骤可能包含流程中的多条路径。然后，要充分测试 ETL 例程，则需要在多种现有优化技术（全边、全节点、全进/出边、全对）中选择一种来测试每个独立的块（算子）。例如，3 个测试案例事实上可能足以充分测试 5 条路径中包含的逻辑。

在上文提及的金融服务公司，经证明，此方法对减少过度测试、缩短测试周期和改进测试质量十分有用。例如，包括用于构建流程图模型的时间在内，该方法只需 40 分钟即可创建 19 个覆盖率达 95% 的测试案例 — 而在之前，150 个测试案例的覆盖率仅为 80%，且过度测试达 18 倍。在另一个项目中，2 小时便创建了 17 个覆盖率为 100% 的测试案例：这对于先前用 6 小时仅实现 16% 的覆盖率有了显著改进。

3) 自动创建执行测试所需的数据

创建测试案例之后，测试人员需要可涵盖 100% 的可能测试的数据才能执行测试。这些数据可直接从模型获得，也可自动创建或同时从多个来源提取。

CA Test Data Manager 等综合数据生成引擎可提供多种方式来创建所需数据，以便使用基于模型的测试推动 ETL 测试。这是因为，除了功能逻辑之外，流程图也可覆盖以系统中涉及的所有数据。换言之，在构建 ETL 规则模型时，可为每个独立的节点定义输出名称、变量和默认值。创建测试案例之后，执行测试所需的数据可随相关的预期结果从默认值自动生成。

或者如果使用 CA Agile Requirements Designer（前身为 Grid Tools Agile Designer），可通过 Data Painter 工具快速创建系统数据。这可提供一系列综合的可组合数据生成功能、种子表、系统变量和默认变量。然后这些资源可用于创建包含每个可能场景的数据，包括“不良数据”和消极路径。由于每条路径都是新的数据点，因此完全可从综合角度创建用于系统地测试 ETL 例程拒绝无效数据的能力所需的数据。

最后，使用自动数据挖掘功能即可在数分钟内找到多个后端系统中的现有数据。此方法通过统计分析来发现数据库中的模式，这样便可提取模式组、依赖关系或不寻常的记录。

4) 快速提供数据，并“匹配”正确的测试

通常首选现有生产数据与综合数据的组合，其中可通过综合生成使覆盖率高达 100%。高效的 ETL 测试的关键是，“黄金拷贝”数据采用智能的存储方式，这样便可并行请求、克隆和交付相同的数据集。然后这可消除数据约束导致的延迟。

智能数据存储的第一步是创建测试集市，数据在该集市中“匹配”给特定测试。每项测试都分配到准确的数据，同时数据匹配给已定义的稳定准则，而非特定项。由于可从测试数据仓库自动检索数据，也可在数分钟内从多个后端系统挖掘数据，因此匹配测试的数据可消除用于在大规模生产数据源中搜索数据的时间。

测试数据仓库是中央数据库，其中，数据与提取其所需的相关查询一起存储为可重用的资产。然后只需数分钟即可请求和收到数据池，而这些数据池与相应的测试案例和预期结果相关联。运行的测试越多，此库变得越大，最后在很短的时间内便可执行每个数据请求。

关键是，数据可同时馈送到多个系统中，并按提供时的样子进行克隆。这意味着，众多源数据库中的数据集可并行提供给多个团队。ETL 测试不再是线性流程，数据约束造成的漫长延迟也得以消除。当模型改变时，原始数据可保持不变，这使各团队能够并行处理不同发布和多个版本。“版本控制”还意味着，ETL 例程的变更会自动反映在数据中，并为测试团队提供最新数据，以便他们严格测试转换规则。

5) 根据规则执行测试并自动比较结果

测试人员具备充分测试 ETL 例程所需的测试以及执行测试所需的数据之后，如果 ETL 测试能够满足不断变化的需求，则还必须使验证本身实现自动化。

使用数据协调引擎

实现此目的的一种方式是使用测试自动化引擎。CA Technologies 可提供能力倍增的数据协调引擎的优势，也就是说，可以提取匹配给特定测试和预期结果的数据并使其贯穿于验证规则。这就是“数据驱动自动化”，例如，在数据协调引擎中创建的测试工具可提取流程图中定义的 XML 文件的每行数据，并将其作为测试来执行。

然后将根据模型中定义的预期结果提供通过/失败结果。此方法可同时自动化测试的执行过程和实际与预期结果的比较过程。测试人员不用再手动将脚本从数据目标复制到数据源，还可避免比较转换产生的每个字段这一艰苦且易出错的过程。

使用 CA Test Data Manager

另外，定义预期结果之后，可根据这些结果使用 CA Test Data Manager 自动创建通过/失败报告。这可自动化手动数据比较，但是 SQL 仍需从源系统复制到目标系统。

首先，使用上文提到的各种技术在源系统中定义综合数据。然后可创建数据池以模拟 ETL 过程，并将数据从源系统复制到目标系统。可能复制有效的数据集以测试数据未被拒绝，还可能复制有错误的数据集以确保拒绝无效数据。通过创建其他表格来存储测试条件和结果，以及创建基于数据池的且包含测试案例和数据条件的表格，可自动将预期结果与实际结果进行比较。然后便可快速识别出任何缺失的或错误的数据。

6) 自动实施变更

通过基于模型的测试进行 ETL 验证的最大优势之一是能够响应高速变化。由于测试案例、数据和需求之间密切关联，因此模型变更可自动反映在测试案例和相关数据中。这种可追溯性意味着，随着 ETL 例程快速变得愈加复杂，测试能够满足需求，不会在应用程序交付渠道中造成瓶颈。

通过流程图建模，实施变更变得与添加新块到流程图一样快速和简单。接着可应用完整性检查算法，以验证模型，并确保已将每个逻辑片段关联到完整流程中。如果使用 CA Agile Requirements Designer，则可通过 Path Impact Analyzer 进一步确定流程图中路径变更产生的影响。然后可自动移除或修复受影响的测试案例，并自动生成保持 100% 功能覆盖率所需的任何新测试。

这可消除手动检查和更新测试所浪费的时间，而且经证明，自动化重复数据删除可使测试周期缩短 30%。在上文的业务情报团队，基于模型的测试为 ETL 过程节省了大量时间。以前当一个 ETL 规则更改时，需花 7.5 小时来检查和更新测试案例，而 CA Agile Requirements Designer 只需花 2 分钟即可完成。而且，在所有测试案例中，实际只有 3 个案例受影响，因此只需更新 3 个案例即可。

如前所述，数据版本控制还意味着测试人员可获得充分测试 ETL 例程所需的最新数据，即使例程改变也是如此。由于测试数据可追溯到模型，因此当需求改变时，变更会自动反映在相关数据集中。数据可在不同发布和版本中并行提供，而进行高效回归测试所需的数据保留在测试数据仓库中。此外还可锁定有趣或稀少的“不良数据”，以阻止其被其他团队用尽，并防止其在数据刷新过程中丢失。

第 4 部分

摘要

使 ETL 测试实现更高程度的自动化，对任何力求持续交付优质软件的组织是势在必行。从根据静态需求手动编写幽灵代码到提取所需数据并比较结果，手动工作在 ETL 验证中仍然占很高比例。基于模型的测试和智能的测试数据管理可用于自动化所有这些任务，同时让众多团队能够从相同数据源并行工作。

基于模型的测试将 ETL 测试工作“左移”，并使大部分工作集中于设计阶段。在该阶段，进行 ETL 测试需要的所有测试案例都可在很短的时间内自动推导出来。提供 100% 功能覆盖率的测试案例可自动生成，并与独立定义的预期结果建立关联。每个测试进一步“匹配”给执行测试所需的准确源数据，如果这些数据存储在测试数据仓库中，则可并行提供给众多团队。

由于测试、数据和需求之间密切关联，发生改变之后，可快速执行重新测试 ETL 例程所需的测试。因此，与必须手动创建、更新和重新运行测试相比，节省的时间很快超过了花在创建初始模型上的时间。基于模型的生成还提供可重用性的巨大益处，测试组件可存储为共享资产，并与测试数据仓库中的数据和预期结果关联。运行的测试越多，仓库变得越大，最后测试新的或更新的 ETL 规则变得和从现有组件中选择一样快速而轻松。

ETL 测试不会再给应用程序交付制造瓶颈，并能够跟上数据驱动型企业成长的步伐。越来越复杂的例程的可测试性得以维持，这样测试便能够处理收集的各种海量数据，而且不会阻碍持续交付优质应用程序。

联系 CA Technologies，网址：ca.com/cn

CA Technologies (NASDAQ: CA) 致力于开发促进企业转型的软件，为其抢占应用程序经济的先机。软件是各行各业的核心。从规划到开发再到管理和安全性，CA 正与全球各地的公司开展跨移动、私有和公共云、分布式和大型机环境的合作，以改变我们的生活、交易和沟通方式。有关我们客户成功案例的更多信息，请访问 **ca.com/customer-success**。要了解详细信息，请访问 **ca.com/cn**。

1 Jean-Pierre Dijcks，*为何需花很长时间来构建 ETL 过程？*(Why does it take forever to build ETL processes?)，于 2015 年 7 月 24 日检索自 **https://blogs.oracle.com/datawarehousing/entry/why_does_it_take_forever_to_bu**

2 Alan R.Earls，*ETL 的状况：提取、转换和加载技术* (The State of ETL: Extract, Transform and Load Technology)，于 2015 年 7 月 21 日检索自 **http://data-informed.com/the-state-of-etl-extract-transform-and-load-technology/**

3 IBM，于 2015 年 7 月 20 日检索自 **http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html**

4 Jacek Becla 和 Daniel L.Wang，《从管理拍字节中学到的教训》(Lessons Learned from managing a Petabyte)，P.4. 于 2015 年 2 月 19 日检索自 **http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/proceedings/**

5 **http://etlcode.com/index.php/utility/etl_complexity_calculator**

6 Jean-Pierre Dijcks，*为何需花很长时间来构建 ETL 过程？*(Why does it take forever to build ETL processes?)，于 2015 年 7 月 24 日检索自 **https://blogs.oracle.com/datawarehousing/entry/why_does_it_take_forever_to_bu**

7 ETL Guru，*用于存储数据验证规则的 ETL 策略* (ETL Startegy to store data validation rules)，于 2015 年 7 月 22 日检索自 **http://etlguru.com/?p=22**

8 Bender RBT，*基于需求的测试流程概览* (Requirements Based Testing Process Overview)，于 2015 年 3 月 5 日检索自 **http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf**

9 Huw Price，*测试案例灾难* (Test Case Calamity)，于 2015 年 7 月 21 日检索自 **https://communities.ca.com/community/ca-agile-requirements-designer/blog/2016/02/24/test-case-calamity**

10 Bender RBT，*基于需求的测试流程概览* (Requirements Based Testing Process Overview)

11 Software Testing Class，*为何应在软件开发生命周期早期开始测试？* (Why testing should start early in software development life cycle?),于 2015 年 3 月 6 日检索自 **http://www.softwaretestingclass.com/why-testing-should-start-early-in-software-development-life-cycle/**

12 datagaps，*ETL 测试的挑战* (ETL Testing Challenges)，于 2015 年 7 月 24 日检索自 **http://www.datagaps.com/etl-testing-challenges**

13 Robin F.Goldsmith，*给有效软件测试的四个提示* (Four Tips for Effective Software Testing)，于 2015 年 7 月 20 日检索自 **http://searchsoftwarequality.techtarget.com/photostory/4500248704/Four-tips-for-effective-software-testing/2/Define-expected-software-testing-results-independently**

14 Jagdish Malani，ETL：*如何处理不良数据* (How to handle bad data)，于 2015 年 7 月 24 日检索自 **http://blog.aditi.com/data/etl-how-to-handle-bad-data/**

15 Philip Howard，*自动化测试数据生成报告* (Automated Test Data Generation Report)，P.6. 于 2015 年 7 月 22 日检索自 **http://www.agile-designer.com/wpcms/wp-content/uploads/2014/10/00002233-Automated-test-case-generation.pdf**

版权所有 © 2016 CA, Inc. 保留所有权利。此处使用的所有标志可能归其各自公司所有。本文档不包含任何担保，且仅供参考。任何功能说明可能是此处描述的客户所特有的，实际产品性能可能存在差异。

CS200-196717_0416