白皮书 | 2016 年 5 月

使用基于模型的测试来推动行为驱动型开发

Huw Price
CA Technologies

ca technologies

目录

第 1 部分

以透明和协作作为推动原则

随着业务和 IT 计划更加一致，将业务需求以直接转换的方式导入技术项目的能力变得势在必行。由于现代组织越来越依赖可为其客户带来价值的软件，IT 团队需要以更快的速度和更低的成本交付可反映不断变化的业务需求的经完整测试的软件。

通过对比，如果软件需求模糊不清，IT 团队就很难正确理解最终用户和业务分析人员对于软件的预期，那么软件为组织提供的价值也会大打折扣。

开发能够完全合理解读需求的软件却发现并不是最初想要的结果，这会令人感到沮丧，也正是这一点推动了 Dan North 提出的“受行为驱动的开发”。不断经历的“混淆和误解”促使他对将自己引入一系列“死胡同”的开发进行了比较，在这种开发中，他常常会感叹“要是有人告诉我就好了！”[1]

考虑到典型 IT 项目过程中面临的挑战，这种挫败感在开发和测试人员中并不鲜见。平均而言，只有 4% 的项目在一开始就有明确的需求[2]，这进而导致了 56% 的缺陷[3]。这些缺陷在生产期间通常较晚被发现，与在需求阶段就发现缺陷相比，需要花费 50 倍的时间来修复缺陷[4]。这样，因需求模糊而导致的返工解释了为什么 60% 的软件项目失败或费用超支，需求不明确事实上导致 82% 的工作花费在纠错上[5]，并且每年浪费的成本达到惊人的 2500 亿美元[6]。因此，North 完全有理由“决定必须能够以直奔主题并避免所有错误传达陷阱的方式提出受测试驱动的开发 (TDD)”。

受行为驱动的开发 (BDD) 受到来自领域驱动开发的“通用语言”理念的影响，致力于在业务和 IT 团队之间培养协作性。考虑到 IT 团队如何需要快速响应不断变化的业务需求，这是一个合适的概念：核心利益相关者和附带利益相关者（例如，关注业务目标和应用程序行为的人，以及实施这些所需结果和行为的人）都能理解的通用语言，并且鉴于其半正式性，通用语言可以随时转译到要开发和测试之系统的逻辑中。

BDD 随后转变成可以如何准确制定“受行为驱动的需求”的问题，这样才能在开发中成功实施需求。本白皮书将讨论这一点。它将检查在业务和 IT 团队之间培养协作性和透明度的这些核心原则如何在开发生命周期中最好地得到实施。

鉴于通常提倡的需求收集流程，它会认为引入形式化建模可通过合适的工具与 BDD 协作。完整的文档编制（与“最少文档编制”相反）必定意味着无法迅速响应变化这一假设将受到挑战。反过来，人们认为形式化建模会引入需求的完整性以及明确性，而这是最大限度提高软件履行核心利益相关者所需行为的可能性的必要条件。

第 2 部分

不完整性

通过上述讨论的 BDD 驱动原则，显然需求必须能被业务和 IT 团队理解，并且可由测试和开发人员作为技术要求来实施。通过与实际从软件中获取直接价值的核心利益相关者进行互动来收集需求，提供希望应用程序实际表现方式的具体例子或真实场景，这样一定能够保证需求真正在业务团队预期的系统中实现。

然而，尽管这种“从外向内”的开发推动从根本上消除模糊情况，但需求的本质以及产生的方式又带来了一些问题，例如，它们对于测试和开发系统的可实施性如何。

如 Llyr Wyn Jones 在“A Critique of Testing（测试批判）”中所述，需求的实施可被视为一个信息转换的过程。信息将采用一种形式，然后输出为另一种形式。例如，开发人员在编码时将需求转换成软件。完整软件开发生命周期 (SDLC) 被认为类似于信息流：用户与业务分析人员一起制定需求和使用案例；开发人员编写代码，然后测试人员手动编写测试案例并测试脚本。在这些阶段中的每一个中，信息被传递并“转译”为不同形式。

基于将需求作为有效信息的理念，Llyr Wyn Jones 参考“不确定性”的概念定义了“模糊性”。他指出，这样将得到不同的信息输出，因为关于指令含义的不确定性自身将导致多种实施需求的方式。同样，当需要假设关于系统的必要信息时，即出现不完整性[7]。因此，这两个因素都会导致“不确定性”，并且更容易造成核心利益相关者与附带利益相关者之间发生误解。

简而言之，不完整性和模糊性一样，都不利于软件按照核心利益相关者期望的方式运行。不完整性更容易在开发中造成误解，而这正是 Dan North 想要解决的问题，因此形成鲜明对比的是导致得到 BDD 的原则。

第 3 部分

受行为驱动的需求中的不完整性

在 BDD 环境中，Jones 所述的信息流出现在“反馈回路”中，在其中制定需求、开发并测试软件，然后将报告传回核心利益相关者予以评估。在这种情况下，不确定性的风险在于，由于软件一直不能按预期交付，需求也在不断演变，因此“回路”将无限重复。

通过基于业务目标和所需行为来规划需求和测试场景，测试重点在于应发生什么。换言之，首要关注愉快路径。在建议用于编写场景的自然语言中可以看到这一点。

如果依据布尔运算符来考虑系统逻辑，则逻辑可全部减少为三个函数：NOT、OR 和 AND，并通过 XOR 组合这三个函数。继而依据 IF … THEN 语句来进行考虑。

例如，Gherkin 更关注“IF”、“AND”和“THEN”。它描述了场景的起始条件和建立子句，以及真实存在（如“IF”）和场景预期结果（“THEN”）的“触发条件”。每一步中的 AND 功能可多次给予或触发。在以此格式收集测试场景时，显然不存在 OR，这一点在 Rspec 使用的备用自然语言中更加清晰，该语言可能依据“When … Then”语句来格式化场景[8]。换言之，此类自然语言不会考虑不存在触发条件时会发生什么。

然而，由此“OR”反映的决策是系统逻辑的基础，每个代表可能完全不同的路径，因此系统需要将其作为测试的一部分突出显示。这种情况对否定路径尤为常见，应占到总测试工作的约 80%。当不存在触发条件因而无法满足“IF”时将会出现此类意外结果，同时正是这些意外结果和异常值最可能导致系统崩溃。

一些人认为 BDD 非常适合识别在设计程序时遗漏的路径，并解释是否会出现以及何时出现意外结果。由于 BDD 毕竟是迭代流程，因此缩短“反馈回路”意味着可以更快应对这些否定路径，因为测试在生命周期中提前了。一些人甚至建议组织应该“接受不确定性”[9]。

然而，如前所述，在测试中检测缺陷费时费钱，实际上较短的迭代通常会变成迷你小瀑布，从而导致测试失败或不执行。缺陷的可观察性也存在更严重的问题。场景的预期结果 (Then) 本意是可验证的条件,并且测试应该确定以下两项：预期结果出现；预期结果是由正确的触发条件导致并由正确的设置条款引发。但是，当测试基于特殊条件时，则无法明确了解引起结果的原因是否为预期原因，而不是因为两个或更多个缺陷相互抵消。

第 4 部分

基于模型的测试和 BDD

为了进一步实现完整性并增大软件按预期交付的可能性，除了“AND”，BDD 还需要引入“OR”的理念。BDD 离散提出的场景和诸如 Gherkin 等语言随后应连接起来，以反映系统逻辑。这样，BDD 中的测试还会考虑在本应触发而未触发时会发生的情况，从而将系统引入否定路径。

编写和执行测试时，测试人员通常会以隐式方式执行形态模型。例如，执行否定测试时，可能会考虑如果不出现触发条件但具有建立子句时会出现的情况。仍然在 BDD 中，如果两个测试场景共用一个步骤，则它们通过“OR”隐式连接。例如，它们可能共用一个触发条件而不是另一个，因此可以决定系统逻辑。

然而，当以特殊方式执行此建模时，测试可能仅覆盖测试和开发人员想到会发生的场景。当以离散、线性单元提出用户案例时，不能反映它们在系统逻辑内的彼此相关性，在 BDD 中将出现这种情况。因此，测试的功能覆盖率仍然较低，通常为 10-20%，因为即使简单的系统也会有数千种可能的输入和输出组合—人脑显然无法记住这么多内容并准确予以关联。

如果 BDD 要求测试人员无论如何都要建模，假如要求的文档编制清晰而完整，则没有理由说为什么无法以系统性的方式做到。如果测试覆盖通过系统的所有可能路径，包括否定路径和意外结果，则必须要具备这样的系统性。

第 5 部分

流程图建模作为 BDD 的一部分

以下方案说明了如何以符合所述 BDD 原则的方式将建模融入开发。

1. 实例化规范。需求可仍然由行为驱动，并衍生自利益相关者关于他们希望系统如何表现的互动，这一点值得注意。可由业务团队以逆向工程方式将需求制定为流程图，也可由核心利益相关者自身来建立流程。由于流程图可作为通用语言，因此后一种方式是可行的。来自 Bloor Research 的 Philip Howard 认为流程图模型能够以测试和开发人员等所需的系统涉及的所有功能逻辑为基础，不会让业务团队感到“苦不堪言”[10]。

   如果开发和测试人员使用现有 BDD 需求，流程则会涉及场景的重叠步骤，从而随后在系统逻辑中帮助做出决策 (OR)。这样做可强制建模者依据系统逻辑进行思考，强制实现完整性，同时 CA Agile Requirements Designer（之前为 Grid Tools Agile Designer）将识别断开的路径并为遗漏的任何潜在场景提供路径提示。因此，将在反馈回路完成之前识别出遗漏路径或否定路径，从而减少了所需的迭代次数。

2. 对场景建模。实际上，用户案例对于测试和开发而言级别过高。因此，在流程图中对要开发和测试的场景建模，这样，场景的步骤会形成流程图的处理块和决策块，并且场景会通过系统的逻辑变成路径。如果对所有场景建模完毕，每个用户案例都将同样被覆盖，因为场景衍生自用户案例。

   场景的所有方面都可转移到流程图中。例如，处理块可能指定利益相关者的角色，介绍“客户如何尝试从 ATM 取款”。通过流程图的路径可能随后反映 BDD 中的单元测试，指定角色、所需的特性以及益处或预期结果。

   由于是组合方法，因此对于每个场景而言，没有从开头到结束的唯一特别路径。多个场景或特性的功能逻辑组件将在流程图的某些部分组合在一起。这反映出使用自然语言（如 Gherkin）从所收集的 BDD 需求导出测试与对系统核心功能逻辑建模之间的重要差异：建模时，单个测试案例

可覆盖多个场景，因为这些场景可合并在系统中作为一个整体，而不是表示为离散的单元。在减少不确定性的道路上，将用户案例合并为完整的系统是一个关键步骤。

例如，将系统建模为单个流程图，则可以将多个场景合并到一个特定“Given”中。此“Given”可为子流程，或为随后将由决策块描述的多个路径的一部分。

类似地，若干个“When”可能以决策块或处理块的形式位于任何给定路径中。多个场景因此合并，因为如果两个或多个场景共用触发条件，则它们将通过相同的块。每个尚在制定阶段的决策可以随后导向一个新路径或一组路径，并且一直继续直至系统完成建模，包括所有否定路径。例如，如果两个场景共用所有项目，只有一个触发条件除外，这两个场景可以遵循相同的路径，直至最后一个决策块，它们将在此处分路。

3. 反馈回路：验证模型。如果系统由用户和业务分析人员建模，则不需要验证模型，并且 BDD 可前进到下一步。在本例中，反馈延迟实际上非常短，这一点将得到说明。

   如果系统由测试和开发人员建模，模型验证亦同样快速简单。在 CA Agile Requirements Designer 中，可以通过使用案例来执行。每个通过流程图的路径都代表一个独一无二的使用案例，等同于系统建模后的测试案例或场景。验证是指经历一组使用案例—以共用的纯文本词汇表示，并作为流程图—以确认系统的设计是否符合预期要求。

4. 反馈回路：确认模型。由于流程图基于系统的功能逻辑，因此可从流程图导出测试案例。如前所述，每个通过系统的可能路径构成了测试案例，并且 CA Agile Requirements Designer 既能自动识别可能的路径，又能针对路径提供的功能覆盖计算准确的度量。

   此外，可消除无效或冗余的测试，并取消重复的可能测试案例，从而以最小的测试案例组实现最大的功能覆盖率。正如指出的那样，多个场景的逻辑可合并，因此可以使用较小的测试案例组来测试一组场景—例如可以使用 3 个测试案例测试 15 个可能的场景。测试优化提供多种算法来识别哪个路径实现最大功能覆盖率。例如，在一个例子中，CA Agile Requirements Designer 将可能测试的数量从 326 个消减到仅仅 17 个，同时仍然实现 100% 的覆盖率。

   保存好这些路径之后，即可将其导出并执行。此操作可以手动进行，从而将链接到测试数据和预期结果的测试案例推进到 HP ALM/QC 之类的测试管理工具。另一种情况是，可将路径导出为自动化测试脚本，并在大量自动引擎上执行。在上面每一种情况下，所用的测试管理工具会生成“实时文档”和报告。

此自动化流程看起来类似于 Cucumber 的流程，不同之处在于反馈延迟要短许多，同时需要更少的反馈回路。首先，自动生成测试案例即不再需要手动定义测试案例—而采用 Cucumber 时，不再需要手动将 Gherkin 转换为 Ruby 中的步骤定义。例如，在一个例子中，CA Agile Requirements Designer 花费了 90 分钟时间来创建 108 个测试案例，实现 100% 的功能覆盖率。此时间包括流程图设计时间，在实践中，如前面所讨论，由于流程图可轻松调整和重用，因此将节省更多时间。

此外，由于为开发人员提供了完整、清晰且经过验证的需求，所以他们更可能第一次就能交付符合预期要求的软件。如前所述，模糊性会导致 56% 的缺陷，而在这方面，CA Agile Requirements Designer 已将缺陷形成情况减少多达 95%。凭借测试案例提供的 100% 功能覆盖率，很有可能在第一次就发现通过开发到测试阶段的任何缺陷。除了在返工方面节约时间，测试执行时间也有望缩短。复制测试通常会减少 30% 的测试，测试优化则可大幅减少覆盖每个场景所需的总测试数。

第 6 部分

快速轻松地响应变化

也许有人并不认为在 BDD 环境中建模流程图是一项非常费时的工作，并且时间可以更好地花在测试迭代的运行和再运行上。

首先，应该注意到，只在测试和开发人员必须自己制定流程图的情况下会出现这种批评之声：如果业务分析人员和最终用户倾向于流程图制定而不是使用 Gherkin 之类的语言，则将不存在批评。然而，即使技术团队必须逆向工程 BDD 需求，花在流程图制定方面的时间一般而言也较少。制定好流程图之后，即使需求发生变化，流程图也可随时重用。这种方式与必要的完整文档意味着无法快速响应变化这一假设截然不同。

在 CA Agile Requirements Designer 中，更改流程图十分快捷简单，用户只需向流程图中添加一个新的功能逻辑即可。之后，将自动识别并修复被打断的测试案例，同时移除重复或冗余的测试。CA Agile Requirements Designer 将随后生成所需的新测试案例以实现最大功能覆盖率。

这将最大程度地发挥所做初始工作—制定流程图—的价值，并且不需要在进行更改时手动检查每个测试案例，避免了时间浪费。在一个组织中，提出更改请求之后，需要花费两分钟时间来调整现有流程图。CA Agile Requirements Designer 能自动识别并修复 3 个受影响的测试案例，而剩余 64 个保持不变。如本例所示，流程图建模可进一步意味着，只有系统受到变化影响的方面将要重测，从而有助于改进反馈回路的效率。

总而言之，建立流程图所花费的时间最有可能被在如下方面节约的时间超过：因不完整和模糊而造成的返工和调试；建立测试案例；执行测试案例；实施更改请求。因此，完整的文档与行为驱动开发的原则并不冲突，而是可以更好地实施。

第 7 部分

摘要

流程图建模提供了一种可将完整文档引入 BDD 而不影响其核心原则的技巧。流程图可直接从利益相关者互动中导出，并且这样更容易被业务和 IT 团队所理解—流程图提供了一种通用语言，用于说明关于测试和开发团队所需系统的所有功能语言。由于测试周期缩短，这样的流程图也有助于减少“反馈延迟”，测试的手动工作也相应减少。

形式化建模增大了软件在第一次就更准确符合需求的可能性，原因在于提供一组清晰而完整的行为驱动需求。这样减少了在任意给定时刻按业务需求交付软件之前所需的反馈回路总数，使得组织可将精力重点放在创新而不是重复迭代和令人沮丧的返工上。正是由于能够快速轻松地响应变化，才能够以持续开发解决方案，从而努力最大程度实现客户价值。

第 8 部分

关于作者

Huw Price 拥有近 30 年的丰富经验，曾担任多家美国和欧洲软件公司的首席技术架构师，并曾为许多跨国银行、主要公用设施供应商和保健提供商提供高水准的架构设计支持。Huw 被 QA Guild 推选为“2010 年度 IT 总监”，他曾投入数年时间专门研究测试自动化工具，并且已推出多款创新产品，重新精心打造了适合软件行业使用的测试模型。目前他经常在一些知名的国际活动上担任演讲嘉宾，他的文章也常见于多本杂志，例如 Professional Tester、CIO Magazine 以及其他技术出版物。

Huw 最新创立的 Grid-Tools 公司于 2015 年 6 月被 CA Technologies 收购。近十年来，大型组织处理测试战略的方法已发生巨变。通过 Huw 充满远见的方法和领导力，企业在测试中引入了以数据为中心的强大方法，并实施了一些由 Huw 自己构思的新概念，例如“数据对象”、“数据继承”和“中心测试数据仓库”。

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。要了解详细信息，请访问 **ca.com/cn**。

1 Dan North，Introducing BDD (2006)，2015 年 3 月 6 日检索自 http://dannorth.net/introducing-bdd/。

2 Chaos Manifesto 2013 (Standish Group: 2013)，2015 年 3 月 7 日检索自 http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf。

3 Requirements Based Testing Process Overview (Bender RBT: 2009)，2015 年 3 月 5 日检索自 http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf。

4 Why testing should start early in software development life cycle?(Software Testing Class: 2012)，2015 年 3 月 5 日检索自 http://www.softwaretestingclass.com/why-testing-should-start-early-in-software-development-life-cycle/。

5 Bender RBT, Requirements Based Testing Process Overview。

6 Kathleen Barret，Business Analysis: The Evolution of a Profession (IIBA: 2013)，2015 年 3 月 5 日检索自 http://www.iiba.org/Careers/Careers/Business-Analysis-The-Evolution-of-a-Profession.aspx。

7 请参见 Erik Kamsties 的“Understanding Ambiguity in Requirements”，在 Engineering and Managing Software Requirements (Springer: 2005) 中引用，第 250 页。

8 有关此方面的示例，请参见 http://en.wikipedia.org/wiki/Behavior-driven_development#Story_versus_specification。

9 Dan North，Embracing Uncertainty (Goto Con: 2013)，2015 年 3 月 5 日检索自 http://gotocon.com/dl/goto-chicago-2013/slides/DanNorth_EmbracingUncertainty.pdf。

10 测试案例生成，2015 年 3 月 6 日检索自 https://www.ca.com/us/collateral/industry-analyst-report/bloor-research-market-report-test-case-generation.register.html。

CS200-196736_0516