通过用例进行需求管理的可追踪性策略

Ian Spence
Leslee Probasco

Rational Software 白皮书

TP 166, 1998

Rational®
the software development company

目录

摘要

在许多用例建模技术的商业应用软件中，用例模型必须与更传统的需求获取技术结合起来，从而提供一个让项目涉及的所有项目干系人均可接受的需求管理流程。本文探讨了组织在采用用例建模技术作为部分需求管理策略时可以使用的可追踪性策略。

简介/背景

可追踪性项

在讨论需求管理时，特别在使用 RequisitePro 此类工具的时候，“需求”这个词丰富的含义常常使人感到困惑。除了通常定义为“需求”的项以外，我们还需捕获和追踪其他许多类型的项的属性以及这些项之间的可追踪性。这些其他的可追踪性项包括问题、假设、请求、词汇表术语、主角以及测试等等。

捕获并追踪这些其他类型的可追踪性项能够帮助我们有效地管理项目需求。

定义：可追踪性项

需要明确地从另一个文本或模型项进行追踪的任何文本或模型项，目的是为了跟踪两者之间的依赖关系。

在 RequisitePro 中，这个定义又可另述为：

在 RequisitePro 中用一个 RequisitePro 需求类型的实例表示的任何文本或模型项。

RequisitePro 本身提供了一个优秀工具，用于定义、捕获并跟踪值、属性以及软件开发所涉及的多种可追踪性项之间的可追踪性链接。

隐含和明确的可追踪性

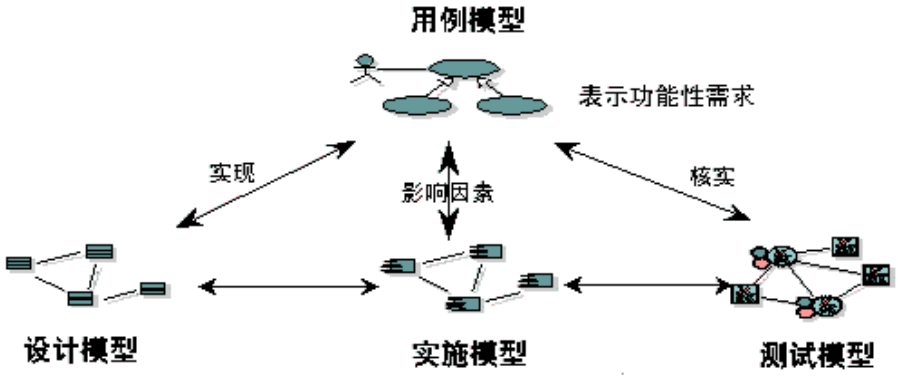
任何开发流程都隐含有一定量的可追踪性。这些可追踪性通常由流程中工件之间的正式关系提供。

这类隐含的可追踪性的示例包括：

- 命名约定
  例如，设计模型中名为 Fred 的类由实施模型中的 Fred 类来实现。
- 构建模型间的映射
  例如，Rose 中的组件视图允许 Rose 中逻辑视图内的包和类显式映射到实施模型中的包。这种映射可以包含对具有不同打包策略的模型与应用程序之间的重命名。
- 模型项本身之间的关系
  例如，在 Rational Unified Process 中，设计模型中的用例实现可追溯至实现它们的用例。
- 确立不同的角度来阐明一个模型中的元素如何满足另一个模型的元素所隐含的需求
  例如，设计模型中的用例实现说明了设计模型的模型元素如何通过合作实现用例的方式。这就提供了一个从用例角度说明设计模型的方法，它可以验证和补充设计模型中类和包的静态封装方法。

所有这些示例都具有一定级别的可追踪性，允许利用开发模型保留的信息执行影响分析。

如下图所示，受用例驱动的开发包括一系列相互关联关系的模型。



该图显示了模型以及它们之间的隐含关系。模型之间的关系具有一定级别的可追踪性，它是开发流程隐含的。

在进行用例驱动的开发时，需要一些支持工件来支持用例模型，并定义完整的软件需求规约。在 Rational Unified Process 中，这些工件是补充规约和词汇表。其他有关的工件还有商业理由以及前景文档，它们包含了项目的需要、目标和特性的定义。

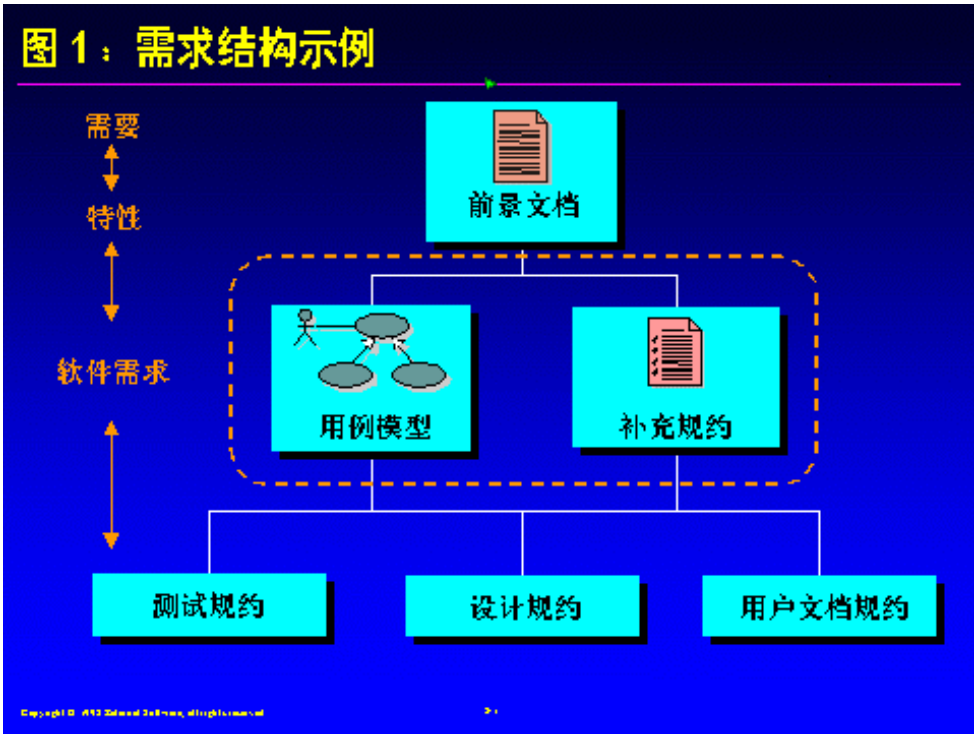
模型间的关系不涉及这些支持工件，因此就不包含在开发流程内建的隐含的可追踪性之中。

隐含关系是开发流程的基础，并且它可以从构建作为开发人员工作的一个有机组成部分中受益。这些关系是建模流程的核心，它们需要在模型趋向成熟的过程中进行构建和维护。

隐含的可追踪性仅限于在建模表示法中可用的关系。那么为何还需要明确的可追踪性呢？

如果我们打算采用需求可追踪性的原则，就需要将需求、模型项及其他可追踪性项集成到一个可追踪性分层结构中。我们最好还要向开发流程添加更多的可追踪性关系。

下图是一个可追踪性分层结构示例，该分层结构显示了前景文档定义的“特性”与用例模型和补充规约定义的“软件需求”之间的关系。它还显示了软件需求如何追溯至测试需求、设计和用户文档。



如果我们仔细查看补充需求（在补充规约文档中定义）与设计、实施和模型之间的关系，就可以发现该关系并未包含在模型间隐含的可追踪性之内。

这是另一级别明确的可追踪性的典型示例，它通常是项目必需的。许多需求由于与用例模型之间存在隐含关系，需要在整个模型系列中进行追踪。补充需求在补充规约里是沿着用例模型获取的，它们不直接与考虑它们的设计模型中的任何包相关，与实现它们的实施模型中的组件也无直接关系。

其他示例包括以下各项之间的关系：

- 系统特性与用例模型之间的关系
- 用例模型与用户文档之间的关系
- 用例模型与测试需求之间的关系

在建立需求可追踪性流程时，需要作出的重大决定之一就是确定所需的可追踪性级别，以及达到该目标需要多大的明确可追踪性。我们希望我们的需求、可追踪性和管理的方案有利于开发流程的执行，而不是使之复杂化或对其施加限制。

可以看到，增加开发工件明确的可追踪性会显著提高项目成本。当考虑到采集和维护这些额外消息的长期成本时，这种影响尤其显著。我们要做的就是为项目确立一个合适的可追踪性级别，使得我们在额外的明确可追踪性上的投资能得到回报。我们的开发人员应该把时间放在开发上，而不是进行追踪。为了达到这个目标，在给项目增加明确的可追踪性成本之前，需要建立可追踪性策略，并对策略进行评估。

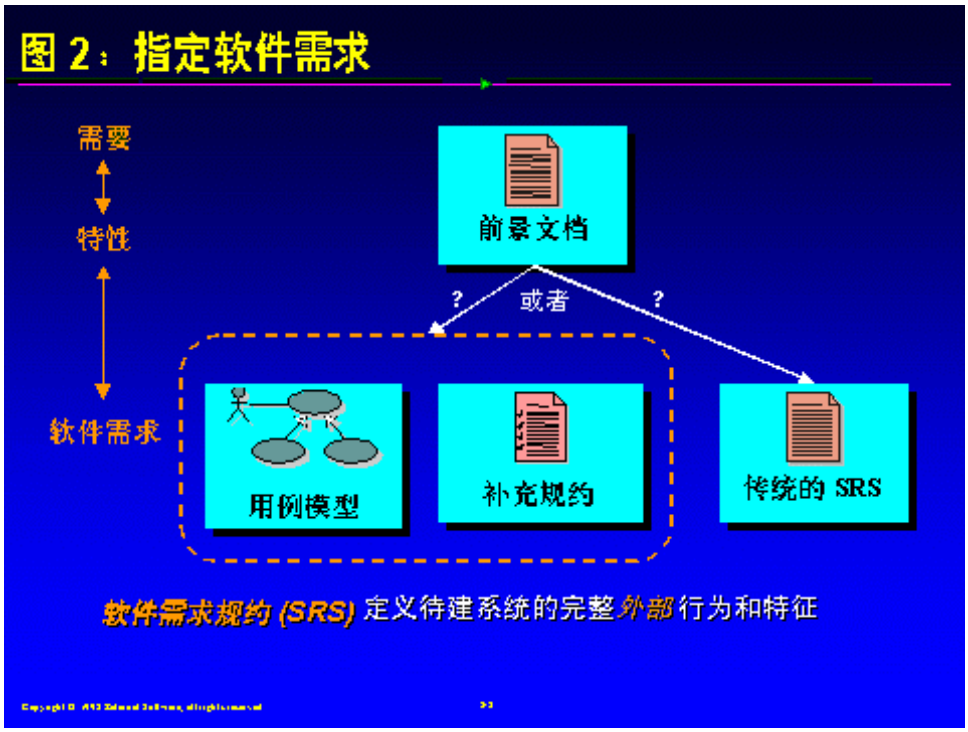
可追踪性策略定义我们希望添加到软件开发流程的明确可追踪性级别。

管理支持工件

上面的图 1 显示了在 RUP 中需求规约所涉及的工件。

这里要注意，用例模型和补充规约形成了完整的软件需求规约(SRS)。这就是说，我们并不需要传统需求管理技术所必需的正式软件需求规约文档。

下面的图 2 显示了传统 SRS 文档通常如何与 RUP 工件相关。传统的 SRS 仅仅是记录软件需求的一种备用方法。这两种方案都可以为我们提供定义待建系统的完整外部行为的软件需求规约，意识到这一点非常重要。



这种关系经常被曲解为这两种需求管理模型不能共存。人们经常以为，在使用正式软件需求规约文档的传统需求管理技术，以及基于使用用例模型及补充规约的需求管理技术的用例模型之间，必须选择其一。实际上，在某些情况下，同一项目中存在两种形式的软件需求管理规约是必然。

可能的可追踪性策略

可接受的可追踪性策略是否不止一个？

有许多可追踪性策略可用于改进需求管理流程，即使在 Rational Unified Process 的框架内，也可能存在多种方案。读者在应用中最常见的策略有四种，分别是：

- 唯用例模型

  在这种情况下，用例模型是系统需求的唯一声明。选择这种方案的项目特点在于，客户和开发员之间有紧密的联系和高度的信任。

用例模型和词汇表以及补充规约形成了系统需求的完整声明 - 不存在其他需要、产品特性或软件需求定义。

- 特性驱动用例模型

  这是 Rational Unified Process 推荐的默认策略。用例模型和补充规约形成了完整的软件需求规约。特性记录在前景文档中，并追踪到用例。如果它们未在用例模型中反映出来，则将它们追踪至补充规约的补充需求。

  在这种情况下，用例模型作为功能需求的主要声明。除词汇表和补充需求外，用例模型和补充需求还用需要和产品特性加以完善。

- 用例模型是对软件需求规约的一种解释

  在这种情况下，用例模型是对正式传统的软件需求规约的一种解释。由于规章、协议或内部原因而被迫采用正式传统的软件需求规约，以及在必须使用用例模型来启用受用例驱动的开发时，经常用到这种方案。

  追踪至正式软件需求规约文档而不是软件需求的(与传统需求管理一样)特性，随后显式追踪至用例模型中。

- 用例模型调和多个传统软件需求集

  用例模型是对来自多个源的软件需求规约集的解释，它提供了单个通用系统的规约。
  在这种情况下，每个项目干系人都有自己的软件特性和软件需求集，这在他们各自的前景和软件需求规约文档内有详细的说明。这些多样化的观点，以及可能存在的相互矛盾的期望，都需要在单个用例模型中进行调和（该用例模型指定了系统将要执行的操作内容）。这一策略在处理独立项目干系人的大集合时非常有效。

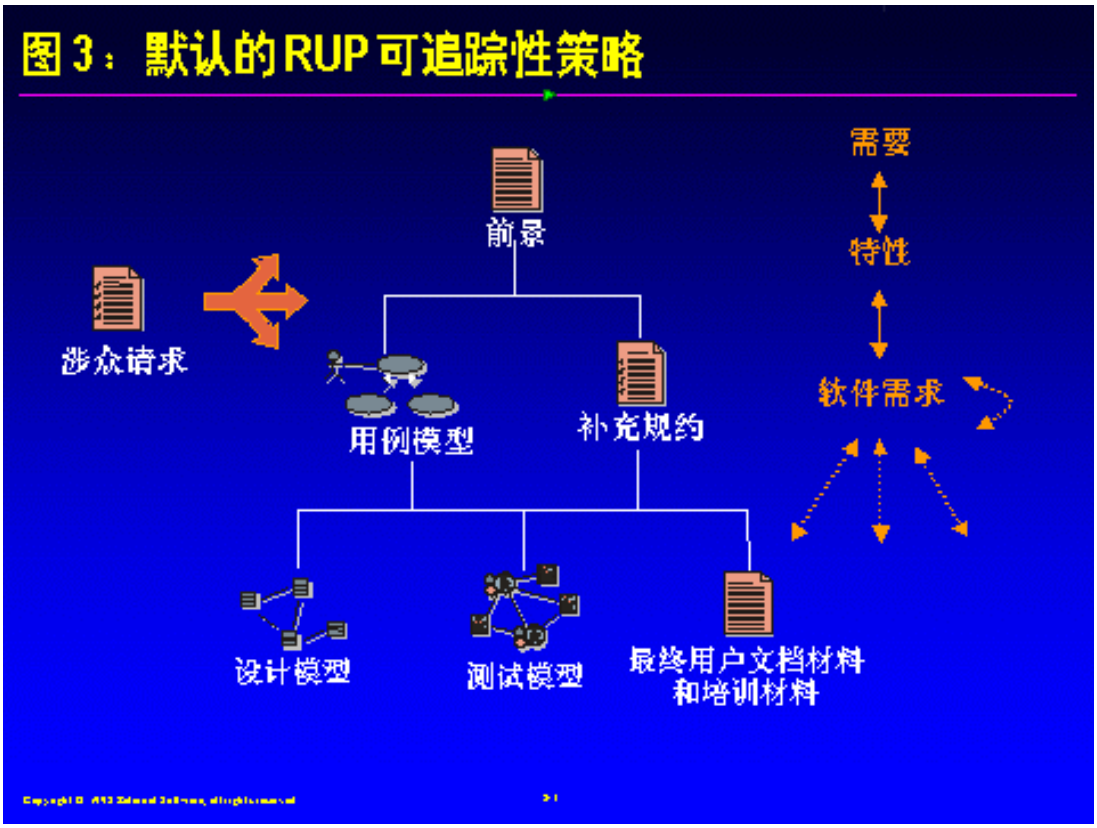
在不包括选项 1 在内的所有情况下，我们把用例模型与传统需求可追踪性流程的元素结合起来。
当然还存在许多其他选项，它们也许根本没有用例模型。我们把这些选项称为“ 无用例模型 ”。

这些方案及其他方案，在下面的可追踪性策略分类中有更详细的论述。

为何要采用混合方案?

可以看到，以上的两种极端方案（唯用例模型和无用例模型）采用的观点极其纯粹，都只允许存在一种形式的需求捕获。两者都假设有一种万能方案可适用于所有的项目和项目干系人群体。两者方案本都看到了成功的希望，但由于它们在处理复杂情形以及“真实世界”项目中产生的项目干系人关系集时表现呆板和无能，最终落得个声名狼藉。

Rational Unified Process 推荐如下的可追踪性分层结构：



这是上面的“特性驱动用例模型”选项，而且也可能是最有效的可追踪性策略。但应该注意，即使 Rational Unified Process 采用了这种方案，它也并不总是最有效的。

将用例建模作为功能软件需求规约的唯一机制，这种做法证明是有问题的，有例为证：

- 存在许多矛盾的需求源（即许多矛盾的期望需要进行跟踪和管理）。
- 承担项目的组织坚持与现有的传统需求捕获流程保持兼容。
- 让项目干系人参加建模专题研讨会，制定一个大家一致同意的需求模型时存在困难。
- 采用用例建模技术在现有需求捕获流程的约束下驱动面向对象的软件开发。
- 项目干系人群体不能或者不愿意直接在用例模型内表达所有的功能软件需求级别期望。
- 客户确定了产品将以传统软件需求集的形式交付。在进行开发投标的时候，这种情况很常见 — 传统的需求声明成为交付合同的一部分。

我们认为，应该采用哪个方案必须根据每个项目和开发组织的具体情况来决定这个问题不存在万能的解决方案。那种企图强行将所有项目都用一个需求管理方案来解决的做法是愚蠢的。

应该记住，Rational Unified Process 是一个可配置的流程，能够处理本文介绍的、不包括“无用例模型”方案在内的所有可追踪性策略（Rational

Unified Process 的用例驱动本质排除了采用该选项的可能）。在编写 Rational Unified Process 开发案例 的过程中，其中决策之一是决定采用哪种方案。

关于可追踪性策略分类

为了定义可追踪性策略，我们需要一种分类和定义策略项的机制：

定义：可追踪性类型
具有共同特征和属性的一类可追踪性项（例如需要、产品特性、用例、软件需求、测试需求、主角、词汇表术语等）。

注意：在 RequisitePro 中，可追踪性类型用需求类型表示。建立可追踪性策略涉及了

三个紧密关联关系的活动：

- 确定定义可追踪性项所需要的可追踪性类型集。
- 确定这些可追踪性类型之间的有效可追踪性关系。
- 确定可追踪性项必需的属性，以便实施有效的项目需求管理。

可追踪性策略分类通过记录已知可追踪性项集以及它们的可追踪性关系，方便了前两个步骤的执行。（它不包括第三个活动，因为可追踪性类型有关属性的定义目前超出了本文的讨论范围。）

在分类中描述的可追踪性策略都使用了同一个基本的可追踪性类型集的子集。

- 需要
- 产品特性
- 软件需求（功能性和非功能性）
- 词汇表术语
- 用例
- 用例段
- 主角

注意：在用例建模的时候，通常唯一的软件需求是补充规约定义的补充需求。

若能在所有传统可追踪性类型与用例模型的构成部分之间进行追踪，项目就有许多可以选择的可追踪性策略。

在可追踪性项之间有两个级别的可追踪性：

- 基本可追踪性

  无论选择哪种可追踪性策略，都要应用一些基本的可追踪性。这些可追踪性隐含在可追踪性类型的本质当中。其中包括如用例与用例段之间或用例与主角之间的关系。

  在阅读以下可追踪性策略段中的概述图时，这些基本的可追踪性不在每个策略中重复说明，但在默认情况下包含在可应用的可追踪性策略中。

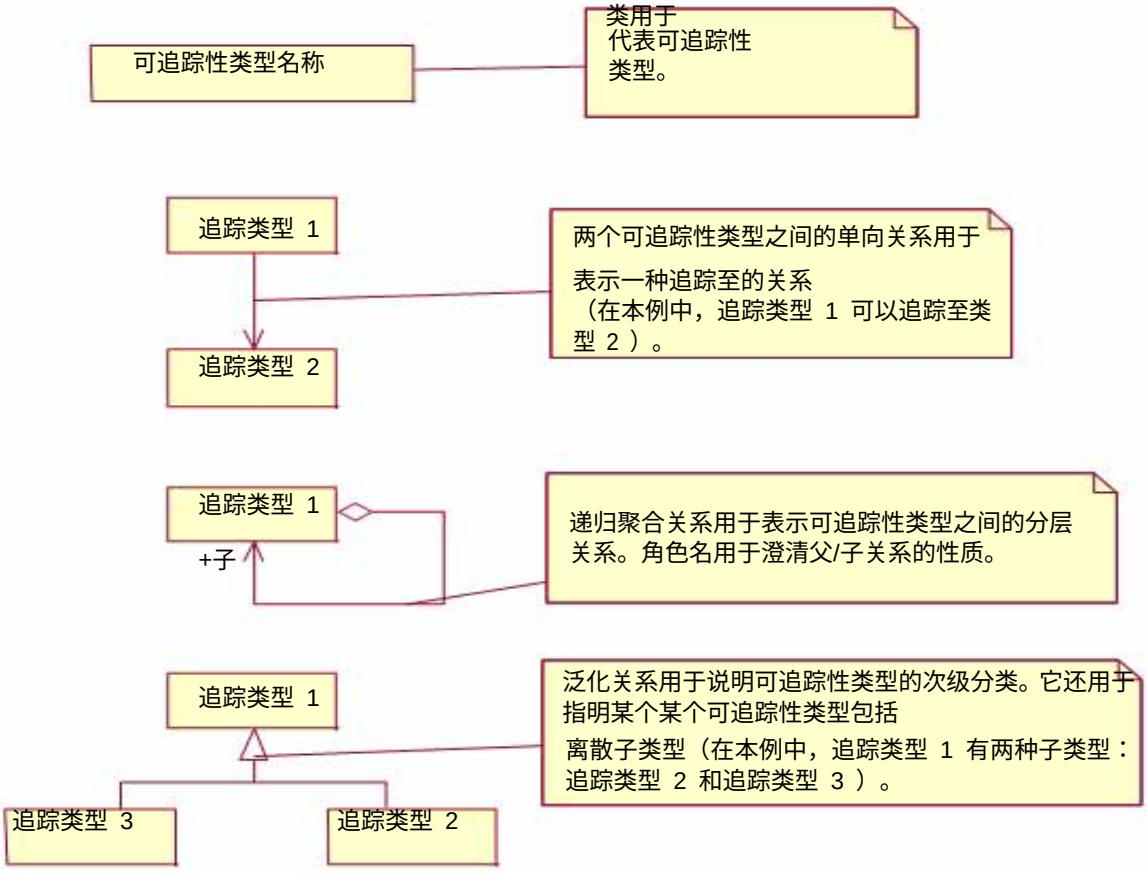
- 扩展可追踪性

  这是为支持特定的可追踪性策略而引进的可追踪性。这些可追踪性更具主观性，而且对不同的可追踪性策略也是不同的。

可追踪性策略分类

图解符号

可追踪性类型及其可追踪性关系显示为统一建模语言 (UML) 图表。下图说明了如何解析在此环境中 UML 的使用。



为了充分理解该图，在实施 RequisitePro 中的定义时了解所用的实施映射是有用的。表解释了图解符号如何映射到 RequisitePro 项目上。

图解符号	RequisitePro 映射
类/可追踪性类型	需求类型
关系	RequisitePro “追踪到”关系
聚合关系	分层需求
泛化关系	通过添加一个附加的“子分类”属性， 对需求超类型进行的分类

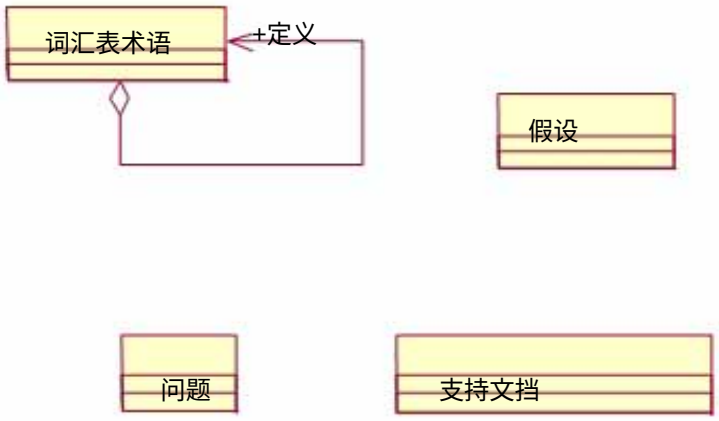
注意 ：RequisitePro 允许将任何可追踪性项都追踪到任意其他项。可追踪性策略定义的是有意义的可追踪性链接，这些链接将成为项目需求管理策略的核心。

支持的可追踪性类型

说明

在本节中，我们定义了一个支持性可追踪性类型集，它可用于支持所选的任意可追踪性策略。

概述



可追踪性类型

可追踪性类型	说明
词汇表术语	这一可追踪性类型定义代表词汇表术语及其定义的可追踪性项。 它包含在支持的可追踪性类型集中，因为不管选择采用哪个可追踪性策略，词汇表都是必需的。
问题	这一可追踪性类型允许添加代表在 RequisitePro 内需跟踪的问题的可追踪性项。随后这些问题与受其影响的可追踪性项联系起来。 追踪与词汇表术语有关的问题就是使用问题可追踪性类型的一个示例。如果定义未确定，或者定义有争议，问题就会出现，并包含在 RequisitePro 中。这就确保了问题不会被遗忘，并允许建立一个视图，报告所有与未解决的问题有关的词汇表项。 另一个使用这个可追踪性类型的典型示例就是，在复审用例和其他开发工件时跟踪出现的问题。
假设	这一可追踪性类型允许跟踪所作的假设。随后这些假设可与受其影响的任意可追踪性项联系起来。
支持文档	这一可追踪性类型允许向可追踪性分层结构添加任何需要的文档。这在将那些预先存在的、用于澄清另一个可追踪性项的意义或目的的示例或文档包含在该类型时特别有用。 RequisitePro 灵活的可追踪性机制允许将支持文档与任何类型的任何可追踪性项关联关系起来。 使用支持文档类型的一个示例就是，将详细的 EDI 消息规约作为词汇表的支持信息，或者作为使用消息的用例附录包含进来。

基本可追踪性

可追踪性链接	说明
词汇表术语到词汇表术语	此关系允许我们使用单个可追踪性类型同时捕获词汇表术语及其定义。
支持性可追踪性类型到其他任何可追踪性类型	这些支持的可追踪性类型可追踪至所选可追踪性策略包含的任何其他可追踪性类型。

无用例模型

说明

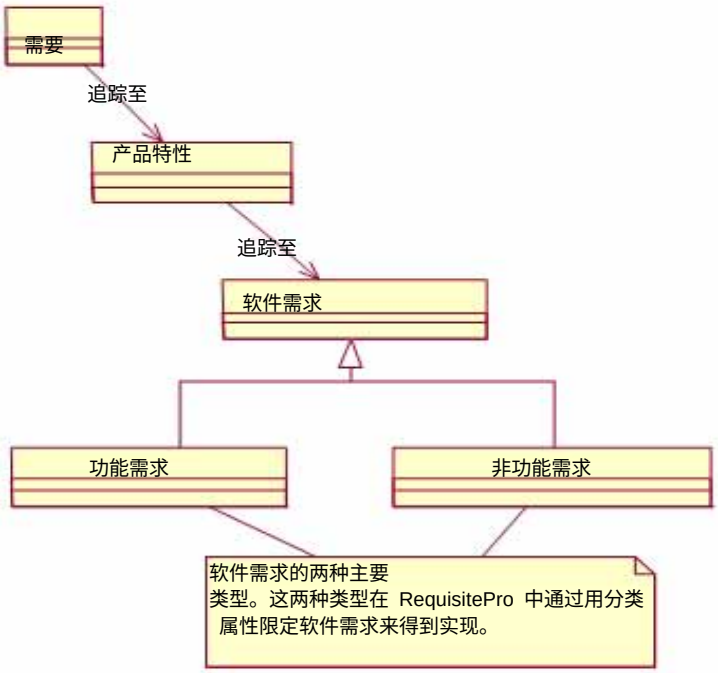
在这种情况下不存在用例模型。需要产生了产品特性，而产品特性依次产生了记录在正式软件需求规约里的软件需求。

“我不能没有讨厌的用例模型！”项目经理们的话一语中的。

特征

特征	值	注释
明确的可追踪性	高	实施需求管理技术但不使用用例的项目往往在可追踪性类型之间维持着高级的显式可追踪性。
信任	低	
可说明性	高	
正式性	高	
完整性	低	很难评估传统软件需求集的完整性。
文挡集	大	衡量文档集通常用英尺而不是英寸。
重点	合同	需求捕获流程的重点在于，在客户和开发员之间建立一个在法律上可强制实施的合同，而不是对有待解决的问题和所提出的解决方案取得一致理解。
可理解性	低	用户群体和开发人员通常无法访问需求文档。需求文档由许多单独的线项组成，线项按类型或者功能范围进行分组，给复审人员的环境提示很少。
流程	典型 瀑布式	传统需求获取技术常常作为瀑布式开发流程的一部分实施。由于缺乏环境以及评估需求的任何子集的完整性存在困难， 这都不利于采用迭代式和递增式的开发流程。
开发风格	功能 分解	在将需求转变为解决方案时，需求按类型或功能范围分组常常导致连续的功能分解。

可追踪性概述



可追踪性类型

需求类型	说明
需要	为了证明购买或使用的合理性而必须 解决的业务或运作问题（机会）。也称之为目标或目的。
产品特性	直接满足某一需要的系统功能或特征。通常理解为系统的“公开优点”。
软件需求	正在构建的软件必须符合的条件或具备的功能。

可追踪性*摘要*

可追踪性链接	说明
需要追踪至产品特性	每一种需要由一个特性集实现。这种关系允许追踪每个特性的业务利益。
产品特性追踪至 软件需求。	每个特性由一个软件需求集实现。 这种关系允许追踪每个软件需求的业务利益，并在产品特性级别进行软件需求规模管理。

优点和缺点
正面：

- 容易理解
- 对法定合同有好处（请看许多与已交付软件满足/无法满足指定需求有关的法庭案例）。
- 是许多标准流程所推荐的。
- 允许有详细的、低级别的正式可追踪性。
- 不会由于引进“惊世骇俗”的新思想而扰乱了现状。

反面：

- 难以完成需求捕获 - 非常容易停滞在需求阶段。
- 难以理解以这种形式表达的需求。
- 难以评估需求变更所带来的影响。
- 单个需求没有相关环境。
- 维护成本高。由于缺乏隐含的可追踪性，项目必须承担维护大量显式可追踪性关系的高成本。
- 由于缺乏相关环境，确定有意义的需求子集难免存在困难。这又使得规模管理以及产品的递增式交付更加难以进行。

示例
需求可追踪性的无用例模型方案广泛应用在许多业务领域的诸多项目中。很多组织都要求有一个正式的传统软件需求规约作为正式合同谈判的基础。这就导致他们认为传统的需求管理方案是适用于项目的唯一方案。

唯用例模型

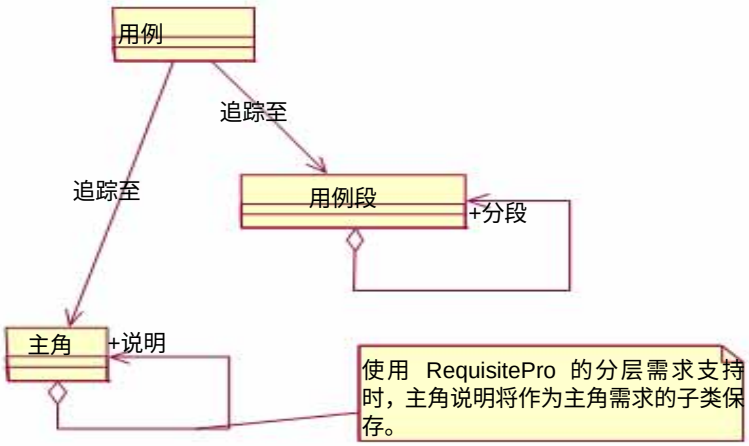
说明
“用例模型是我的需求。”客户和开发人员之间的紧密联系和高度信任通常是采用这种方案的项目的特点。该方案一般用于内部可说明性低的项目。在这些项目中，开发人员希望通过与客户一起开发用例模型（或经用户批准进行开发），展示或取得对需求的清晰理解。

在这种情况下，用例模型是系统需求的唯一声明。用例模型、词汇表和补充规约形成了系统需求的完整声明。

特征

特征	值	注释
明确的可追踪性	低	不要求有明确的可追踪性。用例驱动方案所提供的隐含可追踪性就足够了。可能不维护显式可追踪性，因此不使用需求管理工具。
信任	高	缺乏任何需要或特性级别的分析意味着项目干系人给予用例模型的开发者高度信任，相信他们能交付正确的系统。
可说明性	低	
正式性	低	
完整性	低	尽管用例模型本身有利于建立软件需求规约的完整性，但缺乏到项目干系人需要的可追踪性通常导致生产 出来的系统不完整或者过于精细。
文挡集	小	该方案包含最小的文档集。用例模型表达用户观点。
重点	用户	用例模型容易为项目中的所有项目干系人所理解。
可理解性	高	
流程	典型 的迭代式和 递增式	用例把软件需求放在有利于迭代式和递增式开发的环境中（用例提供一个优秀的交付单元）。用例还可以与瀑布式开发流程一起使用。
开发风格	典型 的面向 对象	尽管用例可以使用任何开发风格，但通常用于驱动面向对象的软件开发。如果不采用面向对象的模式，则要求具有高级别的显式可追踪性。

可追踪性概述



可追踪性类型

可追踪性类型	说明
用例	此可追踪性类型定义代表用例的可追踪性项。
用例段	利用此用例段能够将用例段包含在可追踪性分层结构里 它还允许我们追踪到单个流以及构成用例的其他属性。 子段分层关系的出现允许我们获取各个段的独立部分。例如，它可以让我们确定组成前置条件段的前置条件。 注意 :在某些情况下，确定用例事件流内的单独软件需求是可以的（这种情况下段非常小），但除非用例本身已经稳定，否则不要进行这样的尝试。
主角	此可追踪性类型定义代表主角的可追踪性项。

可追踪性摘要

可追踪性链接	说明
用例至用例段	每个用例由一个用例段集组成。这种 关系允许我们追踪哪些用例段组成了哪一个用例。
用例段至 用例段	一些更为复杂的用例段包含许多子段。例如，事件流可能包括许多子流，而前置条件段则由许多前置条件组成。
用例至主角	这种关系允许我们查看哪一个用例包含 哪些主角。
主角至主角	这种关系允许我们使用单个可追踪性 类型同时捕获主角及其简要说明。

优点和缺点

正面：

- 文档集最小
- 需求管理所涉及的工作量最小
- 很好地支持规模管理、影响分析和递增式开发。
- 用例易于理解。

反面：

- 没有追溯回项目干系人需要的关系。在开始制定解决方案之前没有实际尝试分析问题。
- 许多人认为仅仅基于一个用例模型还难以接受合同。
- 如果不进行任何需要分析，就很难知道用例模型本身何时描述了一个合适的解决方案。在编写用例的时候，想象力容易失控。
- 如果执行定期发布，在没有任何比用例本身级别高的信息的情况下，难以管理产品和持续管理项目干系人期望。

示例

这种方案通常用于小规模的非正式内部项目。在这些项目中，开发人员和用户的合作密切。

用例模型定义产品特性

说明

在这种情况下，用例建模作为主要的需求获取方法使用，并且用例模型成为系统提供的产品特性定义以及软件需求声明。

该方法由于不具有规模调整功能，因此只适用于生命周期短的小型开发项目。即使每个用例代表一个系统特性，但特性仍会多于用例，而且在实际情况中，特性会对

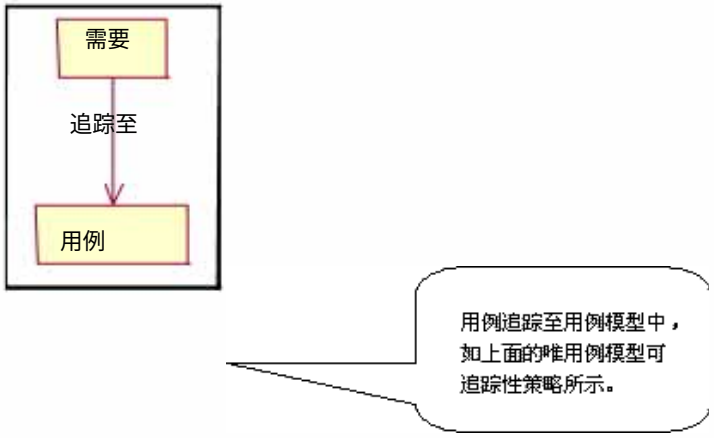
很多用例有影响。随着系统的演进，每个发布版的新特性越来越少，而且以新用例形式出现的可能性更小。

特征

这是前面的“唯用例模型”方案的一个变种。我们在讨论这种方案时只注意了它们之间的少数区别。

特征	值	注释
明确的可追踪性	低	与唯用例模型相同。只存在一小组的需要， 而这些少量的额外可追踪性 仍然产生了对少量明确的可追踪性的需求。
信任	中/高	向用例模型添加需要导致策略的信任程度 比唯用例 模型的稍低。
可说明性	低	
正式性	低	
完整性	中	用例模型本身有利于建立软件需求规约的完整性， 而追溯项目干系人需要的额外可追踪性 则有助于验证用例模型 的适用性。
文档集	小	这种方案包含一个非常小的文档集。包括 用例模型，加上包含需要的前景文档。
重点	用户	用例模型和需要都持用户观点。
可理解性	高	需要和用例模型都容易为项目中的 所有项目干系人所理解。
流程	典型 的迭代式 和递增式	与唯用例模型相同
开发风格	典型的 面向 对象	与唯用例模型相同

可追踪性概述



可追踪性类型

可追踪性类型	说明
需要	其定义见“无用例模型”
用例	其定义见“唯用例模型”

可追踪性摘要

可追踪性链接	说明
需要至用例	在这种情况下，需要直接追踪到用例。假设 用例在产品和规模管理中可以扮演产品特性的角色。

优点和缺点

这种方案与“唯用例模型”策略很相似。除了以下补充说明和警告之外，其余优点和缺点完全相同。

正面：

- 在这种情况下，用例模型与项目干系人需要有关，这有助于评估用例模型的适用性。

反面：

- 在项目的初期阶段，从表面上看用例定义了系统特性，但随着项目的成熟，两个概念将产生分歧。
- 用例不是特性 - 一个从表面上看能节省时间和劳动的策略将很快成为无法维护的废物。

示例

尽管据观察，在小型内部项目内有过使用这种可追踪性策略的尝试，但由于其升级性和长期的产品演进问题，我们不推荐使用。如果用例模型准备用回溯至项目干系人需要的可追踪性加以完善的话，建议采用“特性驱动用例模型”策略。

特性驱动用例模型

说明

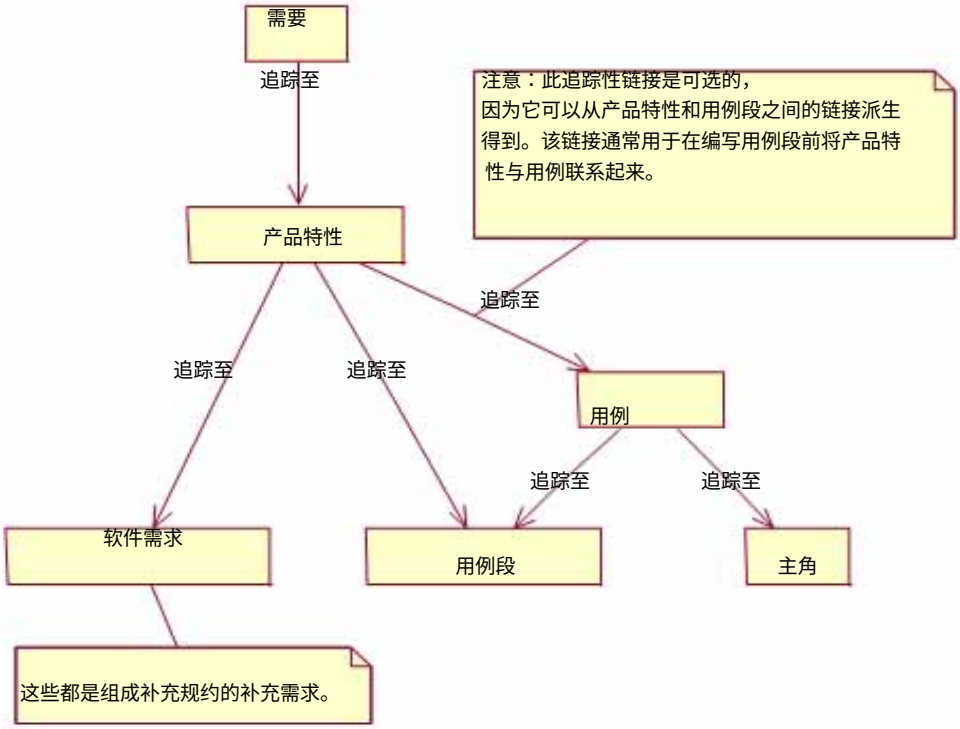
“用例模型和补充规约形成了我的 SRS。” 这是 RUP 拟定并推荐使用的策略。需要和产品特性记录在前景文档中，并追踪至用例。如果它们未在用例模型中反映出来，则将它们追踪至补充规约。

在这种情况下，用例模型作为软件需求的主要声明。它通过补充规约加以完善，补充规约包含了用例本身不易表达的软件需求。

类别

特征	值	注释
明确的可追踪性	中	除了用例模型的明确可追踪性之外，我们必须明确地维护需要、特性与用例模型之间的可追踪性。
信任	中	
可说明性	高	
正式性	中	向用例模型添加需要和产品特性，将产生一个比仅仅维护用例模型更为正式的需求管理 流程。
完整性	高	软件需求兼有特性和用例观点，这使得在获取和区分软件需求的优先级方面就有可能达到高度的完整性。
文档集	中	现在我们有了一个包含需要和特性的前景文档，一个 用例模型和一个补充规约。
重点	用户、 项目干系人和 项目经理	向用例模型添加需要和特性，拓宽了需求活动的重点，使活动更积极地把产品经理及其他所有项目干系人 和用户包含进来。特性是一个管理项目干系人期望的强大工具，它对软件需求的用例观点作了很好的补充。
可理解性	高	需要和特性的定义以及用例模型和补充规约，共同提供了一个容易为项目中所有项目干系人所理解的需求模型。
流程	典型的 迭代式 和递增式	与唯用例模型相同。
开发风格	典型的面向 对象	与唯用例模型相同。

可追踪性概述



可追踪性类型

可追踪性类型	说明
需要	其定义见“无用例模型”
产品特性	其定义见“无用例模型”
用例	其定义见“唯用例模型”
软件需求	适用于整个系统或不容易 适合用例的任何软件需求。其中软件需求 是正在构建的软件必须符合 的条件或具备的功能。
用例段	其定义见“唯用例模型”
主角	其定义见“唯用例模型”

可追踪性摘要

可追踪性链接	说明
需要至产品特性	其定义见“无用例模型”
产品特性到用例	可选的可追踪性链接。产品特性可直接追踪至用例。这一选项允许在编写用例段之前为用例分配产品特性，在产品特性级别对用例模型执行影响分析，反之亦然。
产品特性到用例段*	产品特性追踪至用例段。它允许在特性的基础上管理用例模型的规模，并有利于在比用例本身更合适的级别上进行特性集与用例模型之间的影响分析。 追踪至用例的所有产品特性还必须追踪至用例段之一。
产品特性到软件需求*	产品特性还追踪至补充规约里的软件需求。没有追踪至用例模型的所有产品特性必须追踪到补充规约中 的至少一个软件需求。
用例到主角	其定义见唯用例模型
用例到用例段	其定义见唯用例模型

*每个产品特性必须追踪到至少一个用例段或一个补充软件需求。否则它将不包含在软件之中。

优点和缺点

这种方案最大限度地利用了用例和传统需求管理方案所提供的好处，同时尽量减少它们的缺点。

正面：

- 容易理解
- Rational Unified Process 推荐使用。
- 允许有详细的低级别的正式可追踪性。
- 软件需求持产品特性和用例的观点有利于完成需求获取 - 最大限度地减少停滞在需求获取和捕获活动的机会。
- 软件需求以容易理解的形式表达。
- 此可追踪性策略有利于需求变更的影响分析 — 可清楚理解未实现一个特性或用例段所造成的影响。
- 单独的需求包含用例和（或）产品特性提供的环境。这使得确定有意义的需求子集更加容易。这又使得规模管理以及产品的递增式交付更加容易进行。
- 完整的文档集最小。
- 最大限度减少需求管理包含的工作量。

- 该解决方案的缩放性很好。如果执行定期发布，则在特性和用例级别上都能进行规模管理，从而允许所有项目干系人在任何他们认为合适的详细级别上追踪项目进展。
- 在这种情况下，用例模型通过产品特性与项目干系人需要相关，这有助于项目干系人评估用例模型的适用性。

反面：

- 并非所有组织都接受
- 尽管许多组织已经成功地根据主要作为用例模型表达的软件需求编写合同，但还是有一些人认为很难完成。

示例

这种方案适用于用例被接受作为一种表达大部分软件需求的恰当形式的所有项目。

用例模型是对软件需求规约的一种解释

说明

“用例模型是对正式 SRS 的一种解释。” 当出于法规或内部协议的需要强制实行正式的 SRS 的时候，这种方案最常用到。

在外购或进行定价开发时， SRS 通常被认为是合同取得一致所必不可少的部分。这就导致两种典型情况的产生：

客户为开发组织提供了一个传统的 SRS，作为系统开发的起点。

SRS 文档是项目生命周期初期的强制性或规定的可交付文档。每个项目必须有一个传统的正式 SRS 文档，它用与所有其他项目相同的方式表达系统需求。

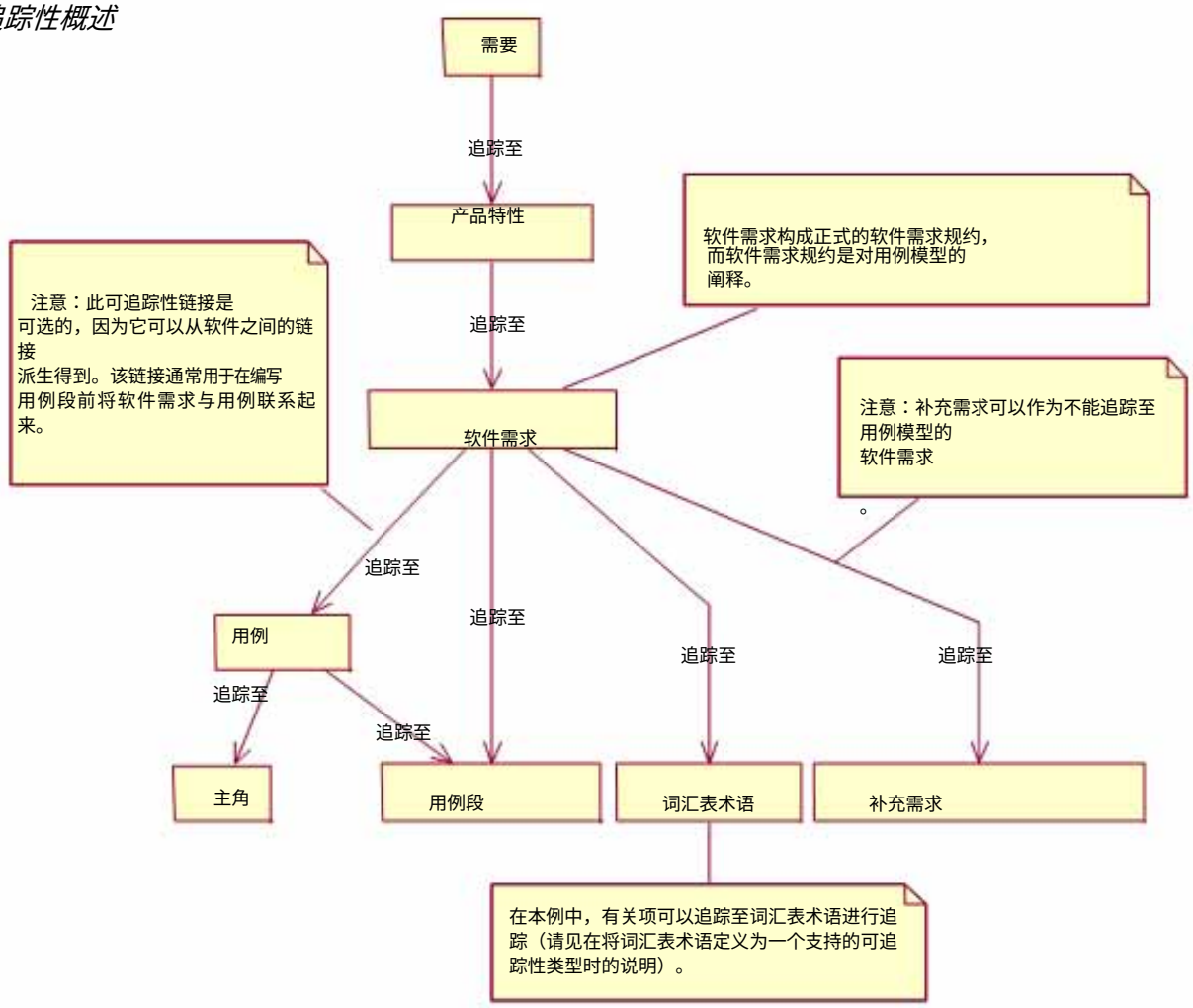
在这些情况下，用例模型用于为项目规模内的所有软件需求建模并重新解释这些需求。采用这种方案时，通常首选 SRS - 当然还有其他技术可以使用类似于传统正式 SRS 的形式提供用例模型保存的信息（特别是在采用“特性驱动用例模型”方案的时候），而不用创建另一个软件需求定义。

注意：采用这种方案，不要求用“传统的”软件需求集作为所需功能的完整声明 - 用例模型将提供 / 确保功能规约的完整性。“传统的”软件需求仅仅用于获取项目干系人直接确定或提出的软件需求。

特征

特征	值	注释
明确的可追踪性	很高	必须具有“无用例模型”方案 要求的所有明确的可追踪性，以便维护正式的SRS。 另外，追踪其他软件需求到用例模型还产生额外的开销。
信任	很低	这是一个“双保险”的方案，意味信任程度很低。
可说明性	高	再次重申，这是一个非常正式的方案，
正式性	很高	并且两个需求管理方案可以并行使用。
完整性	很高	使用用例模型对软件需求规约 加以完善使其成为一种高度 加以完善使其成为一种高度完整的方案。注意：在这种情况下，用例模型确保了系统功能有完整的规约。软件需求规约集不必具备同一级别的完整性。
		在这种情况下，我们基本上是指定了系统两次。
文档集	很大	采用这种方案是为了履行现有的由传统 SRS 表达的合同，或者为了适应现有的要求用 SRS 作为开发者和客户之间的合同的开发 方式。
重点	合同	可理解性 中 在一开始产生两个软件需求定义可能 很容易混淆，但将用例模型作为主软件需求定义应该会使系统需求的说明易于理解。
流程	开放式	虽然用例模型经常被添加到传统的 SRS 中以便促进迭代式和递增式技术的使用，但周围存在的素材足以支持几乎所有的开发流程。
开发风格	开放式	虽然用例模型经常被添加到传统的 SRS 中以 便促进使用面向对象的技术，但周围存在的素材足以支持几乎所有的开发风格。

可追踪性概述



可追踪性类型

可追踪性类型	说明
需要	其定义见“无用例模型”
产品特性	其定义见“无用例模型”
软件需求	其定义见“无用例模型”
用例	其定义见“唯用例模型”
主角	其定义见“唯用例模型”
用例段	其定义见“唯用例模型”
词汇表术语	其定义见“唯用例模型”
补充需求	适用于整个系统或不容易适合用例的 任何软件需求。这些软件需求不必通过原始的软件需求集重新说明，但它们可能只是那些无法追踪到用例模型的规模软件需求。

可追踪性摘要

可追踪性链接	说明
需要至产品特性	其定义见“无用例模型”
产品特性到软件需求	其定义见“无用例模型”
软件需求至用例	功能性软件需求追踪至用例。非功能性软件需求的一个子集也追踪至用例。 这种关系允许根据用例的需求和业务利益对用例模型进行高级规模规划和评估。注意：所有局部功能需求必须追踪至用例或词汇表。如果无法以该方式将它们反映出来，那么它们将得不到实施。
软件需求至用例段	追踪到用例的软件需求必须还能够追踪至用例的用例段之一。 这种关系允许根据对用例提出的需求验证用例的用例段。追踪至用例的所有软件需求必须由用例内其中的一个段实现。双重可追踪性允许我们在考虑所需的用例段之前对用例段进行验证，并为用例本身分配软件需求。 某些段可能没有与之匹配的软件需求。 注意：追踪至用例的所有功能性需求必须还能够追踪到用例的某一个段。如果无法以该方式将它们反映出来，那么它们将得不到实施。
软件需求至词汇表术语	功能性和非功能性需求可追踪至词汇表中的项。对于确定系统所包含的实体的属性和关系的“静态需求”而言，情况尤其如此。如果一个词汇表术语能够从软件需求进行追踪，那么该术语必须用于其中一个用例，否则它不可能被带入设计模型中。
用例至主角	其定义见“唯用例模型”
用例至用例段	其定义见“唯用例模型”
软件需求至补充需求	补充需求可以重新声明适用于整个系统或不能很好适应用例模型的补充需求。另一个备用方案就是将所有未追踪至用例模型的局部软件需求看作补充需求，这样就避免了对它们的重复说明。

优点和缺点

十分坦率地说，这个方案比最前面提到的方案要略胜一筹。它只适用于那些用传统 SRS 来说明的项目， 但这些项目又希望使用用例建模技术来获得对所提供的需求的理解并促进用例驱动方案的使用。

正面：

- 允许有非常详细的、低级别的正式可追踪性。
- 软件需求以容易理解的形式表达。
- 此可追踪性策略有利于需求变更的影响分析 — 可清楚理解
  未实现某个特性、软件需求或用例段所造成的影响。
- 单独的需求包含用例和（或）产品特性提供的环境。用例模型的存在使得我们容易确定有意义的需求子集。
  这又使得规模管理以及产品的递增式交付更加容易进行。
- 在这种情况下，用例模型最终通过软件需求
  和产品特性同项目干系人需求联系起来，这有助于项目干系人
  评估用例模型的适用性。
- 大多数组织都接受（有警告） - 这种方案对所有人都是一样的。这种方案经常在初始用例项目上使用，作
  为并行需求流程的一种形式（即项目同时以新旧两种方式运作），或者用于
  隐藏开发者正在使用用例的事实。
- 在组织首次采用或试验用例时，有助于减少破坏。外界不断看到传统的 SRS 允许使用标准程序和合同。

反面：

- 不好理解 - 许多人对传统需求声明和用例模型的
  共存疑惑不解。
- 同时拥有传统软件需求规约和用例模型
  使您容易在需求活动的两个地方陷入困境。在哪一个应该成为完整的软件需求
  规约的问题上很容易起困惑。
- 非常大的文档集必须要进行维护。
- 副本太多，造成需求管理流程复杂化。由于用例
  直接根据需求变更进行更新，因此传统的软件需求可能就废弃不用。
- 这是一个高成本/高维护的方案。

示例

这种方案对使用用例驱动开发技术的开发公司有用，并且传统软件需求规约是他们的合同的组成部分。用例的引进使得开发公司能够展示它们对需求的理解，并以迭代和递增方式交付软件。

在将用例技术引入一个使用传统需求获取技术但又反对转向用例驱动方案的公司时，这种方案也是一个很有用的策略。在这种情况下，目的在于向开发组织证明用例的价值。在对用例的信心增长以后，逐步淘汰传统的软件需求规约。这是转向“特性驱动用例模型”方案的第一步。

用例模型调和多个传统软件需求集

说明

“用例模型是对来自于多个源的正式 SRS 的解释，并提供单个通用系统的规约。”

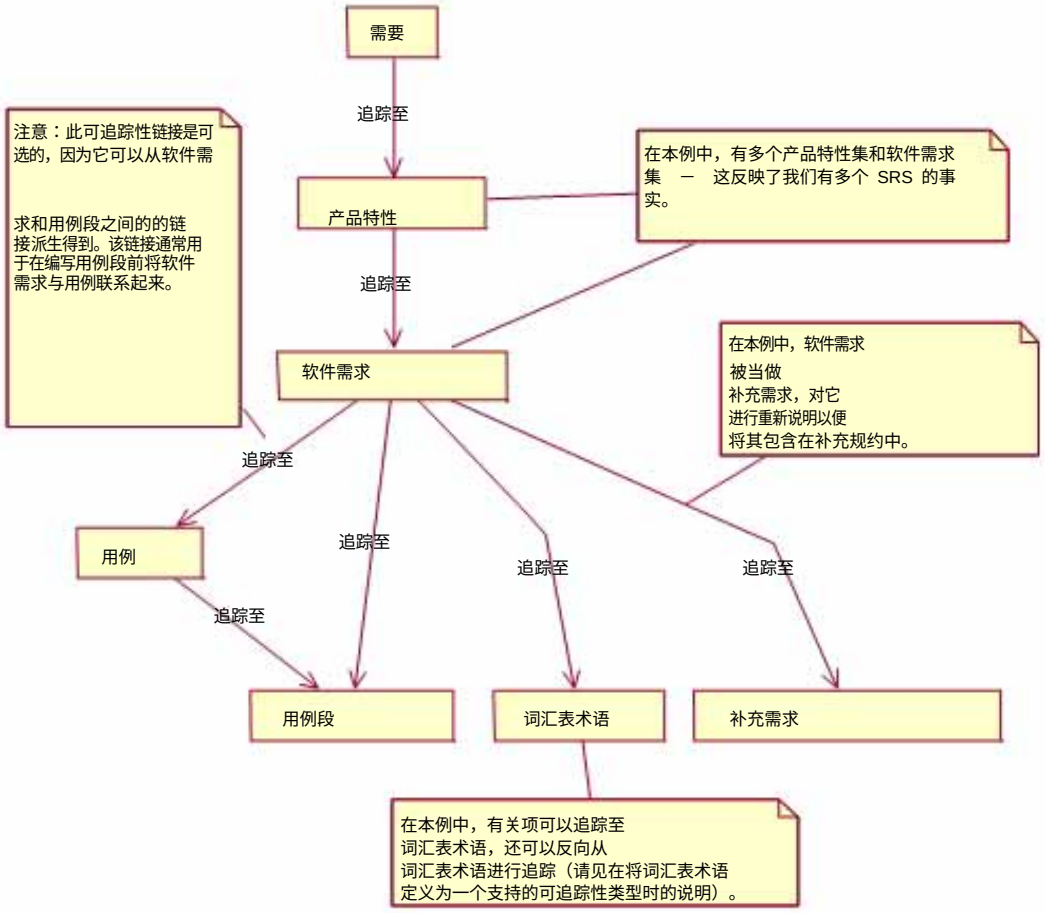
这是对“用例模型是对软件需求规约的一种解释”的另一种说法。除这种情况外，多个独立的项目干系人集提供多个传统的 SRS 解释。在软件公司开发单个应用程序以满足许多不同的、独立的、相互之间没有联系的客户需求时，这种情况经常出现。在这种情况下，用例模型是开发者对系统需求的综合视图，而单独的 SRS 则是独立的项目干系人对他们自己需求的视图（没有集成或者不反映其他项目干系人的需求）。在多个独立需求集和用例模型之间进行追踪允许开发人员评估他们在执行估计各个项目干系人的需求时情况好坏。

特征

这种策略是前面“用例模型是对软件需求规约的一种解释”方案的一个变种。我们在讨论这种方案时只注意了它们之间的少数区别。

特征	值	注释
明确的可追踪性	很高	与“用例模型是对软件需求规约的一种解释”相同。
信任	很低	与“用例模型是对软件需求规约的一种解释”相同。
可说明性	很高	在这种情况下，我们在客户各自的 SRS 中保留了所有独立客户的观点。
正式性	很高	与“用例模型是对软件需求规约的一种解释”相同。
完整性	很高	与“用例模型是对软件需求规约的一种解释”相同。
文档集	很大	在这种情况下，我们拥有所需系统的多个规约，它们在一个用例模型中进行调和。
重点	管理多个 独立的客户	这一方案的重点在于管理多个独立的、可能相互矛盾的需求源，这些需求源在政治、地理或者组织上无法紧密合作。
可理解性	中	与“用例模型是对软件需求规约的一种解释”相同。
中	典型的迭代式和 递增式。	在这种情况下，开发人员使用用例模型作为他们的 SRS。请参见“唯用例模型”
开发风格	典型的面向 对象	在这种情况下，开发人员使用用例模型作为他们的 SRS 去驱动软件开发。请参见“唯用例模型”

可追踪性概述



需求类型

需求类型	说明
需要	其定义见“无用例模型”
产品特性	其定义见“无用例模型”
软件需求	其定义见“无用例模型”
用例	其定义见“唯用例模型”
用例段	其定义见“唯用例模型”
词汇表术语	其定义见“唯用例模型”
补充需求	适用于整个系统或不能很好适应用例的任何软件需求。

可追踪性摘要

可追踪性链接	说明
需要至产品特性	其定义见“无用例模型”
产品特性至 软件需求	其定义见“无用例模型”
软件需求至 用例	与用例模型是对 SRS 的解释相同
软件需求至 用例段	与用例模型是对 SRS的解释相同
软件需求至 词汇表术语	与用例模型是对 SRS 的解释相同
软件需求至补 充规约	在这种情况下，软件需求必须在支持用例模型的补充规约里重新声明，以便为生成一个单一一致的竞争 SRS，从多个独立项目干系人 SRS 汲取开放信息。

优点和缺点

该策略是前面“用例模型是对软件需求规约的一种解释”方案的一个变种，优点和缺点大致相同。

这种方案区别于其他方案的优点在于，它具备让独立的项目干系人以各自正式单独的 SRS 形式处理并保留自己观点的能力。

它也有另一个缺点，即产生了更大的需要维护和跟踪的文档集。

示例

英国的一家软件公司正在开发一个支持保险经纪人的系统，该系统允许保险公司通过电子途径发布新产品。

这个项目涉及 22 个项目干系人，其中约三分之二是经纪公司，三分之一是保险公司。这些项目干系人的需求多种多样，在很多方面，保险公司的需求与经纪人的需求完全矛盾。

在这种情况下，该公司决定为每家项目干系人公司制定一个软件需求规约，详细说明它们具体的需求，使他们能够轻松维护各自的观点。用例模型用于向所有项目干系人介绍系统的综合前景。从初始 SRS 到用例模型的可追踪性能让项目干系人精确看到系统将满足他们的哪些需求，并验证系统是不是适合他们的需要。它还允许软件公司跟踪项目进度，检查是否达到了满足每个项目干系人 80% 的需求的目标。

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