

# 软件隐喻的本质与模式

张逸

www.agiledon.com

## 1 引言

隐喻思维具有普遍性，是人类认知得以深化的前提之一，隐喻是丰富人类语言的有效手段。<sup>[1]</sup>然而，一直以来，对于隐喻的讨论主要限于语言学和修辞学，从而将隐喻狭隘化，局限化，使得隐喻在其他领域中的作用未能得到彰显。美国语言学家莱考夫认为，隐喻不是语言的表面现象，而是一种深层的认知机制，它组建了我们的思维，形成了我们对世界的判断，使语言结构化、系统化，从而具有巨大的生产力。<sup>[2]</sup>这从本质上说明了隐喻的深层涵义，即对研究对象的认知能力。在软件开发过程中，我们就是要从纷繁复杂的需求问题域、设计问题域、实现问题域中进行有效地认知，并给予启发和思考，从而深刻地理解软件开发的过程，理清软件问题域的脉络，以寻求合理的解决方案。

## 2 软件隐喻的概念与本质

隐喻 Metaphor 一词源于希腊语 metephora，其字源 meta 的意思是“超越”，而 pherein 的意思则是“传送”，即“意义的转换”。古希腊思想家亚里士多德对隐喻的定义是：“将属于另一事物的名称用来指称某一事物。”这主要属于修辞格研究的范畴。而莱考夫与约翰逊则突破了该范畴，将其延伸到认知的研究，他们认为：“隐喻的本质是以另一事物和经验来理解和经历一件事或经验。”在语言学中，从认知角度解释隐喻的论点通常被称为认知隐喻，它强调从人类的思维高度来分析隐喻的认知功能。

在软件领域中，隐喻并非简单地用一个名词对软件事物进行类比，它还包含了对类比事物之间的体验与分析，不仅要找出两者之间的类同之处，还要借助人们对隐喻事物的现有理解，通过我们已经熟悉的的活动，将软件开发过程联系起来，从而帮助我们更好地认知软件问题域。因此，软件隐喻的本质是一种认知隐喻。我们可以通过在日常生活中无意识获得的基本隐喻系统，在软件开发过程中，受到关联性的启发和影响，使得主观经验和感觉经验相互匹配，然后通过概念融合而形成具有启示意义和指导意义的软件隐喻。这正是软件隐喻的工作机制。此外，莱考夫还将隐喻划分为三大类：空间方位性隐喻、实体隐喻和结构隐喻。其中，结构隐喻总是以具体喻抽象，借助已知事物与现象去认知其更深层或新的事物与现象<sup>[1]</sup>。从这个角度来讲，软件隐喻则属于一种结构隐喻。

### 3 软件隐喻模式

如果我们认为“所有的隐喻都是建立在两个不同事物的连接之上，如此一来，只要我们有时间的话，我们几乎就可以创造出许许多多数不尽的隐喻。<sup>[3]</sup>”确实如此。即使计算机科学的发展不过仅有数十年的历史，却同样拥有着所有学科中最为丰富多彩的语言。在软件开发过程中，我们会碰到种类繁多的软件隐喻，例如臭虫（bug）、菜单（menu）、视窗（windows）；架构（architecture）、服务（service）、对象（object）；黑盒（black box）与白盒（white box）；瀑布（waterfall）软件模型、迭代（iterative）软件模型……这些软件隐喻无一不是从日常生活或其他科学领域中借鉴而来，根据我们原有的认知，实现一种类比的定义，并逐步发展为软件领域的固有概念。

根据目的性的不同，笔者认为，软件隐喻大致可以分为三种隐喻模式。如果以隐喻的方式给出，即为蓝图模式、地图模式和图示模式。

#### 3.1 蓝图（blueprint）模式

在中文语境中，蓝图一词通常引申为一种构想和计划。它往往从宏观的角度，高屋建瓴地对整个体系进行规划与设计。蓝图模式的软件隐喻往往意味着软件思想的变迁，甚至在有些时候，是这些隐喻推动了软件思想的变革。

例如软件设计思想，就经历了从函数发展到对象，从对象发展到组件，再从组件发展到服务的几次变迁<sup>[4]</sup>。整个发展历程可以从每个阶段的软件隐喻中读出其中变迁的含义：面向过程（procedure）思想，面向对象（object）思想，面向组件（component）思想，面向服务（service）思想。事实上，软件设计思想的每次变迁，都代表了人们对软件开发的一种认知。面向过程思想就是“采用结构化的分析和设计技术来理顺编程中的混乱情况。<sup>[5]</sup>”而这种结构化模型背后的推动力就是“过程”。面向对象思想则以“对象”的方式对数据进行封装、继承与抽象，从而实现软件的可重用性、可扩展性。“组件”则是对象的一种发展，它提供了可交换的、可互操作的二进制组件。而随着分布式系统的大量应用，软件隐喻“服务”则应运而生，通过定义良好的接口和契约，并按照统一和通用的方式实现各种平台之间的交互。服务在内部封装了复杂的业务逻辑和业务流程，以供消费者调用，这与现实生活中的服务概念何其相似。正是“服务”的隐喻，推动了面向服务体系架构（Service Oriented Architecture, SOA）的建立与发展。

或许我们无法断定，究竟是先产生了设计思想，然后为了便于理解，才找到一个合适的软件隐喻来表述它们；还是由于受到日常事物的启发，从而根据对该事物的深层认知，产生革命

性的思想变迁。然而，这种蓝图模式的推动与规划意义却是不言而喻的。

最能体现软件开发蓝图的是另一个软件隐喻——“软件工程 (Software Engineering)”，它来自于工程学领域。按照 IEEE 的定义，软件工程是指采用一种有组织、有纪律、可计量的方式来开发、使用及维护软件，也即在软件领域中对工程学的采用。<sup>[6]</sup> 软件工程的隐喻影响巨大，甚至在一定程度上左右了软件开发的发展方向，因为它将软件开发推向了工程学的范畴。借助工程学中的概念对软件工程予以规范、指导，包括从软件的可行性分析直到软件交付之后的维护工作。正如在 1968 年 NATO（北大西洋公约组织）的会议报告《软件工程》中提出：“我们之所以选择了‘软件工程’这个颇具争议性的词，是为了暗示这样一种意见：软件的生产有必要建立在某些理论基础和实践指导之上——在工程学的某些成效卓著的分支领域中，这些理论基础和实践指导早已成为了一种传统教义。”

与之相对的隐喻是“软件工艺 (Software Craftsmanship)”。在《软件工艺》一书中，Pete McBreen 写道：“用‘工艺学’来比喻软件开发，这可以看成是对软件开发的一次追根溯源。<sup>[7]</sup>” 软件工艺认为“软件开发就是工匠的技艺，它融合了艺术、科学与工程学”。Pete 提出软件工艺的隐喻，并非是要颠覆“软件工程”，而是试图纠正过分强调软件工程所带来的问题。例如，软件工程所假设的“有组织、有纪律、可计量的开发方式”，在现实的软件开发过程中很难做到，因为软件开发不是一个确定过程。

蓝图模式的软件隐喻所指代的含义代表了一种方向、思想与目标，它能起到正本清源的作用。然而，我们不能片面地偏废某一种隐喻，而应全面地看待隐喻的象征意义。因为，不同的隐喻彼此并不排斥，应当使用对你最有益处的某种隐喻组合。<sup>[8]</sup>

### 3.2 地图 (map) 模式

所谓“按图索骥”，地图就是一种指南，一个路标，通过它可以带领我们通向预定的目标。地图模式的软件隐喻具有指引性和启发性，它借助隐喻的象征意义帮助软件开发人员直观形象地理解艰深晦涩的软件概念，这在一定程度上降低了学习曲线和沟通误差。

最为常见的地图模式的软件隐喻是“软件开发生命周期”，它以生物学的概念对软件开发的整体阶段进行隐喻，从而形象地表达了一个具有起点和终点的软件开发过程，正如一个生命的诞生和衰亡。在关于软件开发生命周期的分类中，甚至大量使用了隐喻，例如瀑布模型 (waterfall model)、螺旋模型 (spiral model)、喷泉模型 (fountain model) 以及迭代模型 (iterative model) 或增量模型 (incremental model)。

以瀑布模型为例，隐喻的含义即明确地体现了自上而下、不可回溯的本质。它将软件开发生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护等六个基本活动，并且规定了它们自上而下、相互衔接的固定顺序，如同瀑布流水，逐级下落。迭代模型或增量模型则体现了迭代与增量的特点，它否定了“瀑布”隐喻不可回溯的特质，认为这对需求不断变化的软件开发而言是存在弊端的，它将软件开发看作是一种渐进的过程，这种演进式的开发方式，更利于应对需求的不断变化，也便于架构师驾驭整个系统的体系架构。迭代的隐喻体现了在软件开发生命周期中，“需求定义、设计、实现和测试并非顺序的，而是相互重叠和迭代发生的”<sup>[9]</sup>。增量的隐喻则表现为软件产品以渐增的方式完成。利用增量式开发，每一步增量实现了一个或多个最终用户功能，每一步增量包含所有早期的已开发功能集加上增量的新功能特性。系统在逐步累积的增量中增长，这就使得整个软件开发过程是可控的、可视的。

极限编程 (eXtreme Programming) 的系统隐喻同样体现了地图模式的特征。从项目开发的整体角度来看，系统隐喻就像一个航标，指引设计和重构的方向。<sup>[10]</sup>正如 Kent Beck 的定义：系统隐喻是一种描述，每个人员——客户、程序员和管理者——都能使用它来讲述系统是如何工作的。<sup>[11]</sup>这种描述就是地图的展现，项目组成员依据系统隐喻有效沟通，并在其指引下达到共同的项目目标。系统隐喻能使项目中每个人在系统如何工作的问题上达成共识，有助于拟订一个表示各对象及其关系之间的命名系统；隐喻能够帮助发现有关系统的一些新的知识，它既能反映系统的静态特性，又能反映动态特性，从而反映整个系统的体系结构。<sup>[10]</sup>

### 3.3 图示 (diagram) 模式

图示代表具体而形象的指示，它着重体现事物的细节与实现，正如细致而准确的建筑设计图或者机械设计图。图示模式的软件隐喻表现为一种软件模型、软件规范和实现细节。例如架构模式的分层模式 (Layers Pattern)，其中的“分层”就是一种隐喻，通过将不同功能的对象划分到不同的层次。架构中的每一层都是内聚的，并具有相同层级的抽象。分层模式具有强烈的标识作用，软件设计人员与开发人员通过该隐喻就能够确切地知道具体的解决方案。

在 GOF 提出的设计模式中，大量地运用了图示模式的软件隐喻，例如工厂方法 (Factory Method) 模式、装饰者 (Decorator) 模式、门面 (Facade) 模式以及观察者 (Observer) 模式。工厂方法模式将创建者与被创建者隐喻为工厂和产品，引入了生产线的象征意义。装饰者模式将动态地给对象添加额外的职责隐喻为一种装饰，这种装饰行为形象地体现了职责的叠加与组合。门面模式为子系统的一组接口提供了一致的界面，就像门面一样，只暴露调

用户需要的接口，从而化繁为简。观察者模式引入观察者的隐喻，以表现对象之间的依赖关系。当被观察者的状态发生变化时，观察者能够实时地获得通知并自动更新。

图示模式的软件隐喻通过贴切的修辞，帮助人们理解隐喻事物的含义。例如，对于设计模式的初学者而言，隐喻可以帮助他们直观形象地理解该模式的具体目标以及应用场景。同时，图示模式的软件隐喻还有利于沟通。在讨论设计方案时，如果双方都熟悉这些软件隐喻，就可以省略对具体实现细节的讨论，且不至于出现理解上的偏差。

#### 4 软件隐喻模式的指导意义

三种不同的隐喻模式实际上代表了从宏观到细节、从抽象到具体的发展趋势，对于软件项目管理和项目开发而言，具有现实的指导意义。

利用蓝图模式的软件隐喻，我们可以制定宏观的计划与目标，甚至可以在一定程度上决定公司的决策。例如采用“软件工程”或“软件工艺”的隐喻，就代表了截然不同的用人方式与管理机制。软件工程意味着人海战术，软件企业需要根据 CMM（能力成熟度模型）对软件组织的定义，科学化、标准化、工程化地完成软件项目的开发。而软件工艺则更注重“学徒制”的培育方式，重视软件工匠乃至软件工艺大师在团队中的核心作用，随之带来的是全新的管理方式。

蓝图模式的软件隐喻带来的影响是公司级的，而地图模式的软件隐喻则体现在项目的整个开发过程中。例如我们选择的软件生命周期，会直接影响我们的开发模式和项目管理模式。又如我们对软件本身隐喻的理解，是培植（growing）系统，是系统生长（accretion），还是建造（building）软件？项目合作开发是一场游戏（game），还是一个集市（bazaar），或者像园艺（gardening）一样，甚至像耕田、像捕猎、或像是跟恐龙一起淹死在“焦油坑”里？这些理解都会决定我们对软件开发的態度，甚至决定我们选择的软件开发方式和团队合作模式。

图示模式的软件隐喻决定细节的成败，同时也影响了团队成员的交流方式。如果对这种模式的隐喻缺乏共同理解，就可能带来不必要的沟通障碍。最能体现图示模式的是 UML（Unified Modeling Language，统一建模语言），它通过引入用例图、类图、时序图、状态图等图示，对需求建模、设计建模都起到了统一的规范作用。这完全符合图示模式的隐喻内涵，无疑具有影响深远的指导意义。

#### 5 结束语

软件隐喻描述了软件领域中各种特定的现象和事物。借助这些隐喻，我们能更深刻地理解软件开发的过程。然而，软件隐喻不像算法那样是可预测的、确定性的、不可变化的，它具有启发意义，同时又是随意的。这种特质使得它显得不可捉摸，招致许多人对它的质疑，认为在以严谨著称的软件开发中引入一种修辞手法，会造成混淆视听的效果。究其原因所在，主要在于人们对隐喻的本质缺乏足够的认识，仅仅注意到了隐喻的修辞特性，而忽略了隐喻的认知作用。此外，一直以来，我们缺乏对软件隐喻科学的归纳，从而导致隐喻的混乱无序、错综复杂，无法建立一套行之有效的指导原则。因此，若要合理地运用软件隐喻，使得我们能够借助隐喻的力量推动我们对软件开发过程的认识与理解，其关键是要认清软件隐喻作为认知隐喻的本质，同时通过对软件隐喻的模式划分，并以其作为指导，在软件开发过程中找到或者组合适合于自己项目组的软件隐喻，适当地引申软件隐喻的含义，就能从其蕴含的深刻启发中受益。

### 参考文献

- [1] 杨秀杰, 隐喻及其分类新论, 外语学刊[C], 2005年第3期
- [2] George Lakoff, Mark Johnson. Metaphor We Live By [M], USA: Chicago. 1980-01
- [3] 博尔赫斯. 博尔赫斯谈诗论艺[M]. 陈重仁译, 上海: 上海译文出版社. 2008
- [4] Juval Lowy. WCF 服务编程[M]. 张逸, 徐宁译. 北京: 机械工业出版社, 2008
- [5] Grady Booch. 面向对象分析与设计[M]. 冯博琴, 冯岚, 薛涛等译. 北京: 机械工业出版社. 2003
- [6] IEEE. IEEE 标准计算机词典. 1990
- [7] Pete McBreen. 软件工艺[M]. 熊节译. 北京: 人民邮电出版社. 2004
- [8] Steve McConnell, 《代码大全》, 金戈、汤凌、陈硕等译, 电子工业出版社, 2006
- [9] 黄锡滋, 陈光宇, 增量开发和军用软件质量, 装备质量[C], 2007年第5期
- [10] 李敬华, 胥光辉. 极限编程中的系统隐喻, 南京大学学报(自然科学)[C], 2005年10月
- [11] Kent Beck, Embracing Change with Extreme Programming, IEEE Computer[C], 1998-10