软件开发 形式化方法 需求分析 (22)

计算机科学1997Vol.24№.6

软件开发中的形式化方法*)

Formal Methods in Software Development

90-96 TP311.52

郑红军 张乃孝

(北京大学计算机科学技术系 北京100871)

摘 要 With a view to study on formal methods, this paper discusses a few possibilities and difficulties for applying formal methods to every stages during the process of software development. Then, the paper studies about capabilities, limitations and their causes of formal methods from the point of their theories and applications, also studies about some debates on formal methods. Lastly, based on the discussion above, the paper proposes, from the essences of formal methods, several suggestions and some possible directions, in terms of those suggestions, of the research on formal methods.

关键词 Formal methods. Software development

1 形式化方法

随着软件系统复杂度的不断增长,开发正确、可靠的软件,已成为一个亟待解决的问题。形式化方法是解决此问题的一个有前途、有希望的技术,它建立在严格的数学基础上,其目标是希望能使系统具有较高的可信度和正确性,并能使系统具有良好的结构,使其易维护,能较好地满足用户需求。

"形式化方法"一词虽然一直被广泛地应用,但在不同程度上因理解不同,使其具有不同的含义。一般说来,形式化方法是指具有坚实数学基础的方法,是数学上的综合、分析技术的应用,用于开发计算机控制的系统,经常有推理工具的支持,它可提供一个用于模型设计和分析的严格而有效的途径。从形式系统和复杂问题的本质来看,还未有一个适于全面描述和分析一个复杂系统的形式系统。所以,可以说,一个"形式化方法"并不是系统设计者开发系统时可能选择使用的方法,而只是设计者在此过程中希望利用的一种工具之一。

总体上,形式化方法大致可分为五类[1]:

(1)基于模型的方法。给出系统(程序)状态和状态变换操作的显式但亦是抽象的定义,但对于并发没有显式的表示,如:Z 和 VDM。

(2)代数方法。通过联系不同操作间的行为关系而给出操作的隐式定义,而不定义状态,同样,它亦未给出并发的显式表示,如:OBJ、CLEAR。

(3)过程代数方法。给出并发过程的一个显式模型,并通过过程间允许的可观察的通讯上的限制(约束)来表示行为,如:CSP、CCS。

(4)基于逻辑的方法。有很多方法采用逻辑来描述系统的特性,包括程序行为的低级规范和系统时间行为的规范,如:时态逻辑。

(5)基于网络的方法。根据网络中的数据流显式地给出系统的并发模型,包括数据在网中从一个结点流向另一个结点的条件。如:Petri 网、谓词变换网。

在形式化方法的使用中,其间的区别并不总是那么清楚,有些是结合多种方法的多个方面而形成的混合(hybrid)方法,大多数方法都以集合论和谓词逻辑作为其根基。所以,这些方法在技术上都有一些相似性。不过,在表达能力上,却有着一定的不同,这也是上述分类的主要依据。

形式化方法可以两种不同的方式来使用。首先,可用于生成规范,然后将此规范作为传统系统开发

*)本文受到国家自然科学基金项目和863-306项目资助

的基础；第二，形式规范以上述方式产生，然后将其作为验证程序正确性的依据。在第一种情况，数学将被作为生成规范的主要工具。形式化规范的好处在于：精确、抽象、简明和可操纵。操作可以包括规范的一致性检查、原型的自动生成或通过证明的方法推导出规范的一些特殊性质。在第二种情况，除具有与第一种情况类似的益处，还可以利用形式化方法证明规范及其相应程序的正确性，以表明程序与其规范的一致性，这样，可以使软件开发有可能具有与数学证明同样的确定性。

虽然形式化方法在研究和应用上都取得了很大的进展，也越来越为研究者和软件开发者感兴趣，但在理论上和方法上（哲学上）仍有其自身的局限性，而且对有关使用形式化方法开发大型软件系统和解决特殊问题之适用性及其适用程度等问题还不很清楚[4]。本文基于研究的角度，首先讨论了在软件开发过程各阶段使用形式化方法的可能及困难，进而研究了形式化方法在理论上和应用上的能力、局限性及其产生原因，并由此引发对形式化方法的讨论。最后，对形式化方法的研究提出了几点建议以及所能看到的形式化方法研究的一些可能发展方向。希望本文能有益于形式化方法的研究和应用，能够使形式化方法成为解决软件开发问题的有效途径之一。

2 形式化方法在软件开发过程中的应用

2.1 软件生命周期

软件生命周期涉及从初始概念到软件产品，以及产品的使用和维护阶段的软件开发过程。为讨论方便，我们采用生命周期的一个一般模型，此模型将软件开发的生命周期分为五个阶段：

(1)需求规范—描述系统及其操作环境，特别强调系统与环境间的接口。

(2)系统规范—描述系统输入、输出以及它们之间的关系。系统规范是待开发系统的外部特性描述，不涉及系统的内部结构。

(3)详细设计—描述用于实现此系统的算法及数据结构。

(4)实现—程序源代码，规范的一种可实现映象。

下面，我们将分别讨论在上述五个阶段使用形式化方法的可能及其困难。

2.2 需求分析

需求分析是软件开发过程的第一阶段，它将用户的需求从初始概念转换为需求文档，需求分析的结果应描述系统及其操作环境，而且还应能描述不可计算的系统，需求文档是与用户交流思想的主要基础。在需求阶段使用形式化方法将会更加完善形式化方法已有的益处，如：非二义性、完全性、一致性等，这样，形式化方法中的符号系统将会变得更全面、更完整，不仅能描述功能性的需求，而且亦能描述非功能性的需求。然而，极少有专门面向需求的形式化方法。形式化方法中的符号系统还未将表达能力和直觉结合起来，使得符号系统对用户是可表达的（可接受的），同时又不失去形式化方法所特有的精确性。形式化方法离此还有很大的差距。

2.3 系统规范

系统规范仍属于需求范围。这一阶段主要描述系统而不涉及环境，它给出系统接口的精确定义，主要涉及系统应做什么（WHAT），而不是如何做（HOW）的问题。这对于使用代数规范技术非常有利，它采用输入、输出间的关系来描述系统的行为。在此阶段，可以应用两种可能的形式技术：一是发展代数技术以使其可应用于大型系统的规范（尚未见到代数规范应用于大型系统中的实例），这就要求此技术能将规范模块化；二是可能在技术上找到一条可以减少设计自由度的途径。

2.4 体系结构设计

体系结构设计阶段描述系统的接口、功能、结构的初步实现。体系结构这一概念更接近于面向模型的规范和过程代数，但面向模型的规范缺乏结构性。在此阶段应用形式化方法的主要问题是，没有能够完成需求阶段所有工作的方法或符号系统。目前，形式化方法的使用者必须选择适合其应用领域特点的方法，或使用一种折衷的方法，从不同的形式化方法中找到一个合适的方法来完成此阶段的工作。

2.5 详细设计

详细设计是由体系结构规范出发的精化过程。许多形式化方法都支持"精化"这一概念。精化可以使我们定义和验证同一系统的两个描述之间关系的正确性、一致性。详细设计中的保持结构观点与目前

的精化技术是一致的，传统的精化技术主要应用于顺序系统，也有一些技术支持并发系统，如：CCS。就我们所知，目前尚未有可以同时支持并发和顺序方式的令人满意的形式化精化机制。从实用角度讲，为使形式化方法能够应用于详细设计和精化过程，有必要采用一种折衷的方法，基于一种特殊的基础，研究如何将各种形式的(Formal and Informal)规范联系起来。

2.6 实现

在此阶段，已有大量的关于形式处理的工作，即：将一程序与其规范形式地对应起来。这一技术即是所谓的构造方法，它基于从低级规范推导出程序这一想法，将程序构造与验证统一起来，验证环境是基于与构造技术类似的数学基础，但主要关心程序和规范之间的自动/辅助正确性证明。形式实现技术在顺序程序上应用较广，目前也有对并发程序方面的研究。这一技术的使用代价很高，所以主要用于高精确系统的开发。因为高精确系统中一个很小的错误可能会引起极大的灾难。若要使形式实现技术能广泛地应用，还须对其做较大的改进，以提高其效率，降低其使用代价。

3 形式化方法的能力及其局限性

3.1 形式化方法的能力

首先从理论上讨论形式化方法的能力。规范是用于软件开发者与用户相互交流的主要媒体，它就象一种混合方言，虽然形式化方法使用者具有不同的背景，但他们都将也应该以同一种方式来解释它，这正是用于交流之语言所必备的条件。所以，一个交流媒体应该是清晰的，并且是无二义性的。规范的清晰和无二义性并不等于说它是精确、抽象或简明的，但规范的这五个性质是相互关联的。下面我们讨论一下这五个方面的关系，从中可以看出形式化方法的能力。

二义性问题，相对来讲还是比较容易处理的。形式符号系统具有一定的数学性质，所以其非二义性的基础在于内在的数学结构。越是复杂的数学概念，越是建立在更原始的概念之上，如：集合、命题逻辑。这就是说，形式符号系统可以具有合适的解释，并在理论上足以确保规范的一致性解释。

形式规范由一些精确的定义所组成，因为其符号系统的含义是明确定义的。若将英语作为交流媒体，将很难得到精确的规范，对于其他符号系统，如：结构化方法所使用的符号系统，虽然也是精确的，但其只对结构性具有较强的表达能力，而对功能性则表达能力较弱。所以，使用形式化方法，能够得到比其他规范方法更精确的规范。精确规范的直接益处在于：能减少规范中的二义性和误解释的可能性(危险)。精确是形式化方法或形式符号系统的一个特征，是产生无二义性规范的主要依据。另外，形式规范主要的语用益处在于：可以对形式规范进行较详细的构造性检查，因为对此规范中的具体内容的含义不会有争议，有争议的只是内容的完备性。换句话说，精确有助于确认和交流。

由于使用适当的形式机制，使得规范的抽象性成为可能。抽象是人们处理复杂性的主要智力工具之一，而且通过忽视不感兴趣的部分，更有助于清晰性。各种形式符号系统对于精确、抽象地表达概念具有各自不同的能力，但它们均可用于严密地描述概念，更重要的是，它们比自然语言的描述更严密、更精确、更抽象。规范的抽象、精确及简明性都有助于使规范更清晰，当然，良好的结构也有助于清晰性。在理论上，至于为什么形式化方法不能产生良好的结构还不清楚，但这似乎并不是形式化所固有的。

一般地，形式系统(框架)使得表示一个规范与其相应程序之间的映射成为可能。前面曾提到形式规范的一个很有价值的特性：可操纵性。这就是说，可以在明确定义的规则的指导下，分析规范或对形式规范进行变换。利用形式规范的可操纵性可以证明规范的一致性；可以推导出关于此规范的一些重要结果；还可以验证规范的实现过程，至少可以验证源代码相对于其规范的正确性。更一般地说，有可能将不同级别规范间的验证以及规范与程序间的验证问题简化为形式证明问题。这样，形式化方法就可以提供程序对应其规范的非常高的可信度。所以，可操纵性也有助于确认，并且由这种特性可以得到进一步的抽象(推导出的性质)，同样，也有助于规范更清晰。

形式化方法在理论上的这些能力，使得它在应用上也具有一定的实力。一般来说，比较、分析各种

不同软件开发方法的有效性的确很困难。形式化方法作为一种交流的形式，也许是最有效的，因为它很容易在规范上取得一致，也易于形成文档。形式化方法在实际软件开发中的使用并不很广泛，但在其应用范围内，效果还是可喜的。IBM 的 Hursley 曾经发表通过在 CICS 中使用 Z，使其开发费用降低了9%，同时在错误率上也有可观的改进，尽管所开发软件的形式规范部分只占一小部分，由此可以看出，形式化方法的使用对于改进软件的质量是非常重要的。然而，在实际工业项目中形式化方法的应用相对比较少，而在安全系统软件上的应用实例更少。形式化方法在实际应用中仍有一定的局限性。在理论上，最大的局限性大概与二义性问题有关；在实际应用中，最大的问题与可操纵性有关，大部分是因为缺乏有效的工具。下面，我们将详细讨论这两个问题。

3.2 形式化方法的局限性

形式化方法最基本的弱点或局限性与规范确认问题有关。我们可根据"数学的必然性"由规范开始开发软件，但总是怀疑初始规范的真实性(精确性)。显然，如果能够消除这种疑虑是极有价值的，但证据表明，软件错误的主要来源正是规范。这就意味着，规范所使用的数学工具并不能足以保证规范的"安全"性。更宏观地讲，我们面临着一个权衡问题，即权衡投入形式开发的力量与投入研究验证高阶规范方法中的力量。需要注意的是，可以使用证明技术来辅助确认过程，如：通过由一个规范推导出其安全特性，但这只能简单地缩短形式化与现实世界之间的距离，而并不能消除它。所以，我们不能简单地依赖于形式化机制以取得证明规范的安全性。

第二个主要的局限性与规范的解释有关。对于形式规范，在其数学基础意义下，并不是只有一种解释。软件工程师可以根据计算模型解释；系统用户可以根据系统操作环境中的系统使用模型来解释。这样，二义性问题已不是形式规范在其内部逻辑中存在唯一模型的问题，而是不同领域、不同背景和知识下的各种解释的相容性问题。形式规范比其他相对松散的规范的确二义性问题要少，但这并不能说明在其多种解释下不可能存在二义性问题。这就削弱了形式化方法的能力，但我们并不能因此而否定它。

形式化方法的一个基本问题是，所谓的非功能性需求和性质的规范在目前的形式化方法中很难规范清楚。为了能将形式规范与现实世界联系起来，也为了对规范的解释给以指导，可以对规范中的基本实体和其他基本概念给出较松散的描述。在处理规范时，大部分仍以形式化方法处理，在特殊情况下(非功能性描述)可以采用非形式化方法处理。这样，对于二义性和精确问题，形式化方法的真正局限性在于形式化方法只能减少对规范误解释和错误的机会，而不能消除它们。这就要求形式化方法的使用者应正确地看待形式规范。应认识到，形式规范的目的是增加其可信度，减少软件开发中的错误，但形式规范并不能完全保证其可信及无错误。形式规范的可信度在于对软件产品及其开发过程的理解，如果形式证明具有与形式规范同样高的效率，那么，以形式化方法开发出的产品的复杂度将是可观的。换言之，证明本身是非常复杂的且难以理解。这就导致一个问题：形式证明的使用是能够提高还是降低了对一个软件系统的理解度和可信度呢?这个问题很难回答，因为在回答过程中，难免不受当前程序验证工具的能力问题的影响。对于一个规范或程序，至少其停机问题是不可判定的，因此也就无法形式地证明其是否停机。虽然象这类不可判定问题在实际中并不常见，但形式化方法使用者必须认识到，在形式系统中，有一些问题即使是简单的，也是无法证明的。认识到这一点，对形式化方法的应用相当重要。

形式化方法中，形式规范起着一定的关键作用，但规范的使用者不易接受规范的形式化以及规范中极难理解的名词、概念。象 Z 和 CCS 符号系统中所体现出的数学抽象，使得规范能够简洁和精确，但这类符号系统却对规范的清晰性没有多大好处。有许多人认为，清晰性和精确性是有其对立性的，其主要原因是，在实际应用中，我们过分地依赖于规范的非形式解释，而不是根据其内在逻辑来解释它们，以更好地理解规范之含义。一个与之相关的问题是，形式规范技术往往会有许多基本的数学背景，而这些背景又往往与实际问题没有直接关系。事实上，形式规范并不是真正的精确，因为关于形式化方法的符号系统和语义并未能定义得令人满意，就是 Z 也有很多版本，尽管现在在 IFIP 的努力下，得到一个 Z 的标准，当前的形式方法的研究与发展并不象理想的

那样好，尤其在协调形式化方法的表达能力、灵活性与定义的精确性之间的关系上有一定的困难。

另外一个主要问题是我们对工具及其应用应相信到什么程度。关键问题是对于复杂工具应相信到什么程度，尤其是那些比实际问题还复杂得多的工具，比如，我们使用的编译器和定理证明器。因此，有必要对工具进行测试或证明，但这又产生了一个递归问题：对用于验证验证工具的工具，我们应相信到什么程度呢？所以，形式化方法及其支持工具只是减少了开发的风险，如：通过规范的一致性检查，但它并未消除所有的风险，而且还可能引入风险，尤其是在依赖和使用工具这一问题上。

最后，关于形式化方法的教育和训练问题亦是形式化方法中的主要问题。当然，如果软件开发者愿意接受教育和训练，那么，这一问题是易解决的，但情况并非象想象的那么理想。

4 关于形式化方法的讨论

正是上述形式化方法的局限性，它的反对者对形式化方法提出了一些质疑。例如，怀疑形式化方法的表达能力与支持工具的能力，但这些反对意见无疑有益于形式化方法的研究与广泛应用。值得一提的是，上面提到的关于形式化方法的理论问题也影响着实际系统的开发。尽管早在1976年 Gerhart 和 Yelowitz 指出了形式验证程序的失败之处，但问题的原因在于其证明过程的不适当，而不是证明本身有问题。许多反对者认为形式化方法及技术在本质上就是错误的，它不适于软件开发过程。另外一个问题是，形式化方法的局限性到底对其应用有多大的影响？我们认为，形式化方法的局限性并不影响形式规范本身作为交流和文档工具的价值，而验证、精化问题是一个实质性的问题，我们需要用其他方法来丰富精化的证明，并不是形式化方法本身有什么错误。现在并没有一个充分的精化技术完全支持形式化方法，这一点仍是一个很难的研究课题。

形式化方法是有争议的，其支持者认为形式化方法是对软件开发的一场革命，而其反对者则认为形式化方法不可能有多大的发展。因为大多数人对形式化方法并不是很熟悉，所以就很难断定哪一方是正确的，于是就造成了对形式化方法的一些误解和误用。Hall[9]以其研究和应用形式化方法的事实分析了当时对形式化方法认识的七个误区：

(1)形式化方法能够保证软件是完全正确的。事实是，形式化方法也是可能出错的。

(2)形式化方法的全部即是程序证明。事实是，形式化方法的全部是规范。

(3)形式化方法只适用于安全性攸关(safety-critical)系统。事实是，形式规范有助于任何系统。

(4)形式化方法要求具有很高的数学基础。事实是，规范所需的数学是简单、容易的。

(5)形式化方法增加了开发的费用。事实是，从长远的观点来看，形式规范可以降低开发的费用。

(6)形式化方法对用户来说是不可接受的。事实是，形式规范可以帮助用户理解软件系统。

(7)形式化方法未能应用于真正的大型软件的开发中。事实是，形式化方法每天都应用于工业项目中。

最后 Hall 提出，虽然形式化方法不是"万灵药"，但它是一个有力的工具。Hall 还针对上述七个误区，给出了对形式化方法的七点认识。

随着形式化方法的不断研究和发展，对形式化方法亦有了更新的认识，同时也在软件开发界产生了对形式化方法认识的新的误区。Bowen[8]根据其使用形式化方法的切身经验及对形式化方法更深入的研究，又给出了另外七个误区，并以事实解释了误区的形成原因，及误区的错误所在。这七个误区分别是：

(8)形式化方法延误开发过程。

(9)形式化方法缺乏工具。

(10)形式化方法代替了传统的工程设计方法。

(11)形式化方法只能应用于软件。

(12)形式化方法是不必要的。

(13)形式化方法并没有被支持。

(14)形式化方法的使用者总是使用形式化方法。

其结论与 Hall 基本相同，即形式化方法并不是"万灵药"，而只是一种用于改进系统可靠性的方法之一。

形式化方法最激进的批评者对形式化方法几乎持完全否定的态度[7]。Glass 对形式化方法以讽刺的

口吻提出了四点质疑。我们认为，无论是误区还是讽刺的质疑，无疑是对形式化方法研究和应用的一个促进，并不能因此而"因噎废食"。由这些误区和质疑，也不难看出对形式化方法的一些"幼稚"的认识，以及由此而造成的许多误解。事实上，每一种方法都有其自身的理论背景和应用条件。同样，使用形式化方法亦要清楚其理论背景和应用条件，若将形式化方法应用于文学创作，显然是不合时宜的。形式化方法的使用应限制在可形式化的范围之内。然而，现实世界中，并不是所有的实体都能够形式化的，这就提出了人工智能中的基本问题：常识(common sense)的表示问题，这一问题也是阻碍人工智能发展的重要因素之一。若能将常识形式化，那么，形式化方法的应用范围将能大大地扩大。在反对形式化方法的意见中，有很多是将形式化方法应用于不适用的情况而得到的结论，这也是对形式化方法认识之误区和质疑的主要来源。当然，也必须承认，形式化方法的确有其自身的局限性(如在第3节中所述)，这也是任何一种方法所不可避免的。现在并没有足够的信息和证据用以客观地评价形式化方法和其他方法对软件和系统开发过程的贡献。

无论对形式化方法如何争论，我们有理由相信，形式化本身并没有错。形式化方法在技术上已显示出了可观的实力，同时我们也看到，它也限制了软件开发的范围，有些可用其他方法实现的问题，用形式化方法却无法实现或很难实现。这就要求我们能够"超越"形式化方法，深入认识软件开发过程，对软件开发中的非精确、非形式化领域予以同样的重视，以科学的研究方法来研究形式化方法[8]。

5 几点建议及发展方向

基于上面对形式化方法的分析和讨论，我们提出对形式化方法的几点可能的改进，从而也就确定了形式化方法的一些发展方向。

(1)可重用的规范库及更易接受的符号系统将更有助于形式化方法的研究与应用。如：形式化方法与结构化方法符号系统的结合，即可将形式化方法的严密性与结构化方法的结构性结合起来，使得规范更易理解，更易于交流。在这方面，目前也有一些研究成果，对可重用规范的研究目前较少。当然，这一改进工作并不是短期内可以完成的。

(2)改进形式规范的语法、语义定义的质量，从而可以使得形式化方法更加"稳定"。象 VDM 和 Z 这样的形式化方法，就有许多版本。因此造成了形式化方法的不稳定状况。目前已着手开始对 VDM 和 Z 进行标准化工作了。

(3)加强规范语言中对并发控制和容错处理的表达能力，同时也要使精化技术能够处理这类并发机制和容错。这方面的改进也是长期的研究课题。

(4)对于支持形式化方法的工具的可信度问题，一直是困扰形式化方法发展的重要因素之一，如何度量与提高支持工具的质量亦是一个长期的研究问题。

(5)Bell 实验室的 P. T. Devanbu 在第十六届软件工程国际会议中(1994)指出：目前大多数软件系统的容量和复杂度日益增大，需要对软件开发过程中的各个阶段增强基于知识的描述和维护。基于知识的软件工程(KBSE)研究范式正是形式化的知识表示和推理机制，支持多种软件开发过程。随着 KBSE 的发展，该方式需要在基于知识的系统构造、维护、运行和理解方面增强描述能力和推理能力，而描述逻辑可在终止性、形式化语义和高效的推理过程诸方面提供有效的支持。

在此，我们只给出对形式化方法研究的一部分研究方向，而未能给出关于这些研究方向的研究方法。从总体来说，若要在以上各方面得以改进，完全依赖于形式化方法研究的前人结果是不现实的，必须开辟一条新的研究道路，尽力摆脱前人结果的束缚，这样才有可能超越形式化方法，使得形式化方法得到充分的利用。

另外，值得重视的是，形式化方法的研究与发展，也依赖于其他相关学科的发展，如：代数学、范畴论、逻辑、认识论、认知科学、人工智能等。数学作为计算机科学发展的基础，同样对形式化方法的发展也起着至关重要的作用。认识论、认知科学、人工智能等学科的研究能够促进对软件开发过程、智能型软件的形式开发，也对知识表示等方面有一定的重要影响。正如 Hartmanis 提到的，真正的计算机科学家应是一个全面的人才，而不应局限于其特定的领域，应有较强的理解力。同样，研究形式化方法也应考虑其相关学科的发展及其影响。

6 结论

形式化方法的研究涉及许多方面，本文不可能将这些方面全部阐述清楚，有许多文献已对形式化

方法的其他方面进行了讨论，如：文[2,3]对形式化方法的使用提出了切实可行的十点建议，并指出若依此十点建议来使用形式化方法，将会体会到形式化方法的巨大益处。Craigen 等(1995)对形式化方法在大型软件开发中的应用作了一个客观的论述，其中有形式化方法应用于各种领域的十五个应用实例，对每个实例详细地叙述了项目的开发者、开发目的、开发内容、形式化方法在开发中的使用、项目产品的使用、开发中所使用的工具等。Liu 等(1995)对形式化方法在特殊领域中的应用方法及效果做了阐述。另外，本文也未对形式规范语言和具体的形式化方法及技术，如：VDM、Z、B 等进行讨论，这些在文献中均有详细讨论。

对形式化方法近三十年来的研究和应用，已取得了一定的成果，但也应清醒地看到形式化方法在研究和应用中的困难和局限性，更重要的是，应注意对形式化方法研究的研究方法。对形式化方法的研究，我们认为，首先应认识到方法本身并不能解救我们，但它可以帮助我们。也就是说，要认清方法在软件开发中的地位，认识到新的方法和工具可以帮助我们，但它并不能解决根本问题。在这一认识的基础上，对软件开发过程的深入及实质性理解，对形式化方法的研究也具有很大的影响，试图通过形式化软件开发过程来研究形式化方法，我们认为是不可取的，因为软件开发过程本身是人的一种具有创造性的智力行为，在此过程中有智能的因素在起作用，如：人的直觉、常识等因素。另外，在研究模式上也应加以改进，目前的研究模式基本都是研究-推广式。这种模式实际上不符合人的认识过程，人类的认识过程是实践-认识-再实践的反复过程。因此，理想的、正确的研究模式应该是由真正的软件开发者在实际开发中提出问题，提出对形式化方法的要求，然后由研究者研究解决这些问题的可能性及方法，再将其应用于实际的软件开发过程中。这样，对形式化方法的研究就应有强大的经济支持及应用背景，以能使形式化方法充分发挥其应有的作用。当然，目前国内的研究状况很难达到上述要求，甚至在发达国家中也不多见。因此，一种可行的办法是，研究与开发并轨同行。研究者可以暂时不考虑其研究内容在今后是否有什么应用前景，就象纯数学的研究那样，同时开发者亦应不断地提出问题，以此来刺激研究者不断地纠正方向，使其研究渐渐地靠近应用，随着科学及其应用的不断发展，研究与应用必然会走到一起而合二为一。这时，将是科学的又一大进步。这也是科学发展的一种较现实的途径，对于形式化方法的研究也不例外。

最后，讨论一下关于形式化方法的使用和适用性问题。对于这一问题的争论，实际上来源于：到目前为止，还没有足够的证据及实例能够充分证明形式化方法可以在大型软件开发中发挥其效益[5]。当然，必须承认，形式化方法的研究者们的确在某些方面夸大了形式化方法的能力。形式化方法在理论上的能力是很可观的，也是清楚的，但对于其局限性，就有些难以阐述清楚。所以，最好是刺激形式化方法的使用以充分显示其价值。无论是研究还是使用形式化方法，都应认识到，软件工程是一种人类行为，所以，形式化方法不可能保证软件产品的正确性。如果能够愿意从失败中吸取教训，愿意利用现有的最好的技术来检查我们的工作，通过适当的测试和工具，那么，我们将能成功地使用形式化方法来开发高质量、高可信度的系统。总之，形式化方法的研究和使用，具有光明的前途，但也有其不可避免的曲折，任何一门学科的研究、发展都是如此。

主要参考文献

[1] L. M. Barroca et al., Formal Methods: Use and Relevance for the Development of Safety-Critical Systems, The Computer Journal, 35(6), 1992

[2] J. P. Bowen et al., Ten Commandments of Formal Methods, IEEE Computer, Apr. 1995

[3] J. P. Bowen et al., Seven More Myths of Formal Methods, IEEE Software, July 1995

[4] H. Ehrig, Theory and Practice of Software Development, EATCS, No. 37, 1989

[5] C. Floyd, On the Relevance of Formal Methods to Software Development, LNCS 186, TAPSOFT'85, 1985

[6] A. Gulton, Logic as a Formal Method, The Computer Journal, 35(5), 1992

[7] R. L. Glass, A Theory About Software's Practice, J. Systems Software, Vol. 28, 1995

[8] R. L. Glass, Beyond Formal Method, J. Systems Software, Vol. 29, 1995

[9] J. A. Hall, Seven Myths of Formal Methods, IEEE Software, Sept. 1990

[10] P. Loy, The Method Won't Save You, Softw. Engi. Notes, 18(1), 1993