# 设计模式对软件质量的影响:理论在哪里?

软件工程师是习惯的创造者。在软件开发过程中，他们在架构、设计和实现程序时一再遵循相同的模式。1974年，亚历山大在建筑中引入了这种模式，20年后，由于Gamma等人的工作，他们在软件开发方面取得了进展。促进软件设计模式，使程序设计更加“灵活、模块化、可重用和可理解”。 然而，十年之后，这些模式、它们的角色以及它们对软件质量的影响还没有被完全理解。然后，我们开始研究设计模式对不同质量属性的影响，并在2008年第12届欧洲软件维护和再工程会议（CSMR）上，发表了一篇题为“设计模式对软件质量有积极影响吗？“。十年后，这篇论文在2018年第25届国际软件分析、进化和再工程会议（SANER）上获得了最具影响力的论文奖。

在这个奖项的回顾论文中，我们报告并反思我们和其他人关于设计模式影响的研究，讨论关于设计模式的一些重要发现。我们还从这些研究中后退一步，重新检查设计模式在软件开发中应该扮演的角色。最后，我们概述了一些关于设计模式的未来研究工作的方法，例如，识别开发人员真正使用的模式，解释模式影响的理论，或者使用它们来提高编程语言的抽象级别。

## 1 介绍

开发者和大多数人类一样是生物和习惯的创造者。在开发软件系统时，他们会一次又一次地遵循相同的模式，并为类似的问题重用相同的解决方案。在开发和维护活动期间，他们还会寻找软件构件中模式的出现情况，以反映其他开发人员的体系结构、设计和实现选择。然而，这些模式、它们的对象以及它们对软件质量的影响还没有被完全理解。

有大量关于人类行为习惯和模式的文献。在不涉及卡特尔主义者和实用主义者之间有趣的争论的情况下，我们指出，有几个理论试图解释人类行为中习惯和模式的形成、使用和变化，以及这些习惯和模式变化的危险和有用性，例如，理性行为理论，社会认知理论，人际行为理论，或计划行为理论。在1974年，Alexander在体系结构中介绍了这样的体系结构模式，20年后，由于**Gamma**等人的工作，它们进入了软件工程。在软件工程中，设计模式编码“良好实践”，指导开发人员解决反复出现的设计问题。他们提倡使设计更加“灵活、模块化、可重用和可理解”。 Gueh ' eneuc提出了一个理论，试图解释开发人员如何依赖模式来收集有关程序的知识。这个理论和其他有趣的证据支持编码模式的重要性，无论是编程语言特性、重复出现的编程习惯，还是开发人员使用ide的模式。（IDES、集成开发环境） 它们还支持这些模式的作用，以提供软件工程师之间的通用词汇表和教授软件工程师最佳实践。因此，它们支持提高软件开发和软件系统整体质量的模式。

然而，在2008年，我们发表题为“设计模式对软件质量有积极影响吗？”模式对软件质量的影响很少被研究，也没有被完全理解。很少有研究调查设计模式在软件开发活动中的作用，有迹象表明一些模式实际上可能对软件质量有害。因此，我们调查了研究群体，以了解专家对设计模式的看法及其对不同质量属性的影响：Gamma等人提出的四个属性。，即灵活性（可扩展性）、模块性、可重用性和可理解性，以及六个其他有趣的属性：简单性、可学习性、通用性、运行时的模块性、可扩展性和健壮性。

我们的调查显示，专家认为设计模式并不总是提高质量，一些设计模式对Gamma等人强调的属性有特别负面的影响。本论文在2018年第25届国际软件分析、进化和再工程会议（SANER）上获得了最具影响力的论文奖，“因为它开创性地检验了设计模式与代码质量之间的关系”。 自从我们的调查发表以来，研究人员一直在研究设计模式对软件过程和系统质量的影响。例如，Garzas等人通过对JHotdraw的300多个版本的分析，研究了“设计模式是否可以简化对UML（统一建模语言）设计的理解和更改”，而Hegedus等人则研究了“设计模式”对可维护性的影响，JHotdraw是一个设计使用了许多设计模式的框架,并报告了数字之间的相关性设计模式实例和可维护性。

为了鼓励行业从业者和研究人员思考模式对软件开发活动的影响，我们在2013年联合创办了模式促进和反模式预防(PPAP)系列研讨会。研讨会持续了四年，是研究人员讨论模式对质量影响的进展和问题的论坛。它汇集了多达20名参与者，并最终在PPAP'16（与SANER'16位于同一地点）进行了卡片分类练习，强调了社区遇到的三个层面的问题：

1)需要一个强大的(er)社区，在期刊上有专门的模式专刊，(更多)关注模式的研讨会和一系列会议，以及在线审查活动和资源。

2)需要将研究活动与行业实践联系起来，特别是要将经过科学验证的结果推广给从业者，并研究影响从业者正确使用的因素。

3)对教学模式的支持，特别是对缺乏经验和时间限制的学生，通过使用模式、暑期学校和共享教材来审查代码。

在这篇回顾性的文章中，我们回顾了这些以前的研究、事件和它们的结果，并从这些模式的研究后退一步，重新检查模式对软件开发活动的影响。本文不是对模式文献的系统回顾，而是有助于我们理解模式的讨论。在选择、讨论和总结第二和第三节中关于设计模式的作品之后，我们将介绍第四节中文献中存在的许多其他类型的模式。我们对这些模式进行命名和定义，并提供重要的参考资料。模式类型的多样性支持这样的观点:开发人员是习惯的产物，其他研究领域的理论适用于软件工程师。它还强调了系统地、彻底地研究这些模式的重要性，以评估它们对软件开发活动(尤其是软件质量)的影响。我们在第五部分讨论了我们对支持模式的统一原则的理解不足。最后，我们建议研究界应该推广模式研究的最佳实践。我们在第六和第七节中概述了未来研究模式的一些途径。

## Ⅱ鉴定作品的方法

为了调查关于设计模式的文献并确定研究团体所处理的不同主题，我们首先必须协调研究人员在报告关于设计模式的研究时 使用的术语。我们进行了如下的领域分析:

1）识别与设计模式相关的论文集。

2）选择一个这样的集合并下载所有文件。

3）阅读所有论文，确定与设计模式相关的术语。

4）编译所有术语，包括同义词和同音词。

5）对于一个术语的每次出现，识别其同义词和同音词并保留最重要的术语。

6）定义保留术语。

7）将所有术语与保留术语对应起来

这个过程类似于用不同语言编写成公共元模型的逆向工程程序。程序是文献中的论文，而编程语言是每篇论文中使用的与设计模式相关的术语。与通常的逆向工程过程（即建立一个“独立于语言”的元模型，并将不同的程序映射到其中）不同，我们没有定义这样的元模型，因为它只是一个英语语言的模型，所有的术语都是用它来表达和定义的。我们简单地定义一个模型(一组术语)并将所有现有术语集映射到这个集合中。

本节的其余部分将详细介绍域分析的不同步骤

步骤1、2和3:术语的识别：流程的前三个步骤用于标识与设计模式相关的所有术语。在第一步中，我们确定了两个主要的关于设计模式的论文集，可以在线获得。这两个集合在很大程度上是重叠的，并且是其他这类集合的代表。其中一位作者为第二个论文集贡献了60多篇论文。因此，为了简单起见，我们选择第二个集合。

所选的集合包含99个引用。我们下载了59篇PDF或HTML格式的论文，不包括书籍，因为它们的可用性和广度有限，还有一些论文是因为它们的编码不适合搜索。在这59篇论文中，我们识别了所有与设计模式相关的术语，并将它们与相关的引用存储在一个表中。为了同质化，我们把所有的英语术语都转换成单数名词。我们省略了一些术语，它们要么是其他术语的同义词，例如“Definition”被用作“formalization”的同义词，要么对映射几乎没有影响，例如“Detection”vs“身份”，或已商定的定义，如“角色”。

步骤4、5和6：术语定义：流程的步骤4到6涉及保留术语的排序和定义。首先，我们再次研究所有的论文，并将每个发现的术语与使用该术语的论文联系起来，而不考虑其定义。表一在其左栏中列出了42个相关术语及其定义论文的参考文献。然后，我们将我们理解为具有类似定义的术语分组。在这个分组中，我们保留了16个术语。表I以粗体显示保留的术语及其主要同义词。这里不显示同音词，因为我们在步骤7中研究它们。我们选择保留词的依据是一个词的“流行度”（使用它的论文数量）和它在最近文献中的“趋势”。因此，当一个以上的术语流行或最近时，这种选择有时是随意的。然而，表I是有用的，因为它可以帮助社区对每个定义保留的术语做出明智的选择。

步骤7:所有术语与保留术语之间的映射: 最后一步是将文献中的术语与先前步骤中保留的术语进行映射。理解这些论文需要一个共同的背景。表II以列显示了文献的所有术语，保留的术语以行显示。因此，列显示术语的同音异义词，行显示保留术语的同义词。

表二显示了一些有趣的事实。首先，有几个术语是同音词，在文献中重复使用，定义不同。“设计模式”一词尤其如此，它被用来指“设计主题”、“设计模式”、“习语”、“实例”、“发生”和“解决方案”。**：**“设计模式”这个术语的广泛使用并不令人惊讶，因为这个概念无处不在，但它的定义却很松散。以前的作者常常含蓄地使用“设计模式”一词，因为他们的工作范围在当时还没有完全理解。

其次，同一作者对同一定义使用了不同的术语。例如，在Eden的论文中，术语“设计主题”和“形式化”被用来指“设计主题”。 虽然在被研究的论文中很少出现这种情况，但如果不仔细阅读，这样的同义词会导致对作品理解的混乱。最后,表II显示了一些更 专门化 的术语,例如那些描述设计模式的术语 基本模式 或 混合模式 ,不太容易出现同音异义词和同义词,因为它们的作者从一开始就对它们进行了更精确的定义。从表二中我们还可以观察到，一些术语只在很短的时间内被使用，例如“Leitmotiv”，而另一些术语则更为普遍，例如“正规化”。 此表突出了社区面临的词汇表问题，并允许使用一组常见的术语更好地理解有关设计模式的文献。使用保留的术语，我们现在检查论文的内容，以确定研究社区使用设计模式解决的重复使用和主题。

## Ⅲ 软件开发中的设计模式

总的来说，我们确定了七个主要主题:知识共享、开发工具、形式化、正向工程、逆向工程、文档、质量。我们发现许多论文讨论的主题不止一个。

1. 与设计模式共享知识（建立设计模式库）

设计模式是强大的知识共享工具，因为它们封装了开发人员的经验，并提供了跨域通信的通用词汇表。然而，过度使用设计模式很可能导致设计中很难识别参与设计模式的结构。这个问题称为跟踪问题。Agerbo和Cornils分析了设计模式在程序中的使用，并提出了设计模式库的思想来解决系统中跟踪设计模式的问题。

我们同意保持对现有模式的跟踪是很重要的，并建议建立一个存储库，对已经通过实验证明的模式进行审查，以使软件工程师和系统受益。这样的存储库和审查过程将有助于教育软件工程师使用适当的模式，并确保模式继续形成一致的词汇表。这样的存储库还可以减轻在系统中过度使用设计模式和有太多模式可供选择的风险(也由Agerbo和Cornils提出)。事实上，Agerbo和Cornils认为，如果模式大量增长，软件工程师将越来越难以掌握它们，它们将不再形成一个通用的词汇表。这一困难在第四节所示的目前正在使用和研究的许多不同类型的模式中尤其明显。

1. 使用设计模式进行开发

设计模式可以作为开发工具用于不同的软件开发活动:

1)设计:识别问题并提出解决方案。

2)实现:实例化一个设计主题。

3)维护:记录并提出重构建议。

软件开发过程中设计模式的好处已经成为许多论文的主题。贝克建议模式产生架构。类似地，Ram等人提出了通过模式开发系统的面向模式的技术（Pattern Oriented Technique），作为设计模式的协作。另外，在系统维护期间，软件工程师经常识别代码和设计气味，并寻找重构或模式来移除它们。这些重构和模式转换被记录下来，以便将来重用。Lange和Nakamura证明了模式可以作为程序探索的指导，从而使程序理解过程更加有效。其他论文关注开发过程中的设计模式。通过模式执行的轨迹，他们表明，如果模式在认识过程中的某一点被识别，他们可以帮助“填补空白”，并在进一步探索系统，从而提高其可理解性。

其他的论文关注于设计模式的选择，并试图在开发过程中画出规则来结合这些模式。Gueh ' eneuc等人提出了一个推荐系统，以帮助软件工程师进行选择Gamma等人设计的23种模式。Ram等人提出了设计模式组合和应用的规则。尽管进行了这些研究，仍然没有设计模式的语言，即模式语言及其支持工具。在开发过程中使用设计模式的过程也是一些论文的主题。

1. 设计模式的形式化

形式化涉及提出设计模式（或其部分）的规范与指定设计模式并验证其实现的方法。这些方法包括用于在系统中选择和引入设计模式的规范语言和工具。它们防止了与模式意图不一致的实现，这将使这些模式的好处失效。

尽管规范语言很重要，但在文献中却鲜有报道。LePUS是一种功能完备的逻辑语言，具有良好定义的语义。它有一个紧凑的词汇表，可以表示类、函数和继承层次结构之间的规则和关系。类似地，Hedin提出了一种基于属性语法的方法来形式化设计模式，它提供了通过声明性语义规则描述约定的属性扩展。Smith等人还提出了基于&演算的设计模式形式化，并在SPQR工具中实现，以对系统设计中的规则性进行建模和识别。然而，这些规范语言很少用于研究和实践，流行的集成开发环境中也没有工具支持这些语言。

1. 用设计模式推进工程

设计模式对于系统的设计是有用的。许多研究都集中在向前工程环境中的设计模式上。我们将这个主题分为两个子主题:代码生成和语言。

a)代码生成： 随着人们对模型驱动工程（Model Driven Engineering）越来越感兴趣，设计模式可以成为系统设计和自动生成源代码之间的桥梁。存在许多代码生成技术:Mens提出了模型转换技术的分类，并讨论了它们的适用性。Budinsky描述了自动化设计模式实现的工具的体系结构和实现。工具的用户为给定的模式提供特定于应用程序的信息，工具从中自动生成所有模式指定的代码。Soukup探讨了自动化模式实现的问题和可能性。他指出了三个基本问题：在实现过程中丢失模式、使用多个模式导致的相互依赖类的大型集群，以及缺乏具体的可重用模式库。关于这个主题的另一篇论文是阿格博和科尼尔的作品。

b)语言：为了简化设计模式的规范并提高系统的可读性，人们探索了语法扩展和扩展语言结构。编译时间表为实现设计模式提供了一个总体框架。它们表明，软件工程师可以使用MOP创建一个基于语法扩展和扩展语言构造的可重用模式库。类似地Chambers也在这个主题上提出了一个有趣的贡献。Bosch指出了与使用传统的面向对象编程语言实现设计模式相关的四个主要问题。他用分层对象模型layoum解决了这些问题。layoum是一种面向对象的语言为设计模式提供显式支持。它可以通过对其他模式的抽象进行扩展。

1. 具有设计模式的逆向工程

在这个主题中有两个主要的趋势:设计模式检测和基于模式的组件恢复，我们在这里将其视为子主题。从一开始，就有人提出设计模式可以在逆向工程中发挥中心作用，例如，检测系统中出现的设计模式可以提高它们的理解。发生的事情也会改进文档。在系统的逆向工程过程中，设计模式也被用来重构代码气味和设计缺陷“远离”代码。这一子主题已经发表了许多论文。关于基于模式的组件恢复的子主题，我们可以引用Keller等人的工作。

1. 用设计模式编制文档

这个主题包括研究设计模式在系统文档中的作用的论文。许多论文提倡这样一种观点，即记录模式有助于理解设计，从而简化维护，因为开发人员经常试图在系统中恢复未记录的设计模式。它们说明了模式的组合如何能够简单地传达设计的本质，例如，在JUnit中。他们提出了设计模式对文档和设计信息恢复的影响。Lange和Nakamura再次表明，设计模式可以指导探索并简化理解。Hedin提出的方法还讨论了系统文档的设计模式识别。Soukup提出了一种构建可重用公共模式库的方法。还有其他有趣的方法与用设计模式编制文档相关.

1. 设计模式对质量的影响

Ampatzoglou等人研究了实现一些设计模式的类的稳定性。他们报告在设计模式中只扮演一个角色的类比扮演零个或多个角色的类更稳定。这一结果证实了Khomh等人的发现，他们发现在设计模式中扮演多个角色会使类更容易改变。Khomh等人也注意到，在设计模式中扮演多个角色的类与只在设计模式中扮演一个角色的类相比，内聚性较差，耦合性更强，更复杂，更容易出现问题。

Ampatzoglou和Chatzigeorgiou还研究了状态、策略和桥接设计模式对代码质量的影响，发现它们可以提高内聚性、减少耦合和复杂性。Ali和Elish回顾了关于设计模式对软件质量影响的文献，并报告说设计模式对可维护性、演化和变更倾向产生了负面影响。被调查的研究不同意设计模式对故障倾向的影响。

Khomh研究了代码气味和设计模式在系统中同时出现的情况，发现一些设计模式可以减轻(甚至逆转)代码气味对变更倾向的负面影响。

### Ⅳ类型的模式

在软件工程中引入设计模式之前，已经存在很少的不同类型的模式，特别是编程的习语，但是模式确实成为了伽玛等人的书的主流。从那时起，许多不同类型的模式被研究团体定义和研究。在下面，我们尝试详尽地列出并简明地定义这些不同的类型。我们只考虑与编程相关的模式，因此不考虑管理、再造等模式，它更多地与软件开发过程相关。

* Gamma等人引入的设计模式描述和命名了面向对象编程中常见的设计问题及其解决方案。
* Pree定义的元模式是可重用的面向对象设计，使用与领域无关的术语和符号。
* JNI习语捕获Java和C/C++代码之间的常见交互。
* 异常处理习语报告了处理异常的良好实践，特别是在Java中。
* Kpodjedo等人引入的进化模式（提交、突变）描述了系统进化中的模式。
* Mylyn流行的开发人员模式描述了开发人员在ide中的行为。
* 文件编辑模式在源代码更改时捕获开发人员的文件编辑。
* 编程语言特性是模式的一种形式，习语确实可以解决特定的编程问题，例如try with resources习语。
* API使用的模式组合在一起，系统地为给定API显式地设置方法调用及其参数值。
* EJB模式和其他此类专门模式是对API使用模式的补充，并使用特定的框架描述设计和体系结构选择。
* 日志记录模式描述了在何处、何时以及什么内容登录系统以提供有用的信息。
* Gil和Maman引入的微模式是系统中容易自动识别的设计决策。
* Denier等人对继承模式进行了描述，以捕获与继承相关的面向对象编程中的常见实践，并识别违规行为。
* Wuyts等人引入的规则模式是通过使用SOUL来强制设计“不变量”来体现和实施的设计约束。
* 云模式描述了实现基于云的系统的良好实践。
* 体系结构模式是设计体系结构时反复出现的问题的解决方案。
* Petrillo等人提出的调试模式描述了开发人员在调试系统时的行为。
* 当定义并指定为模式时，UML工件可以部分地从源代码中恢复。
* Firesmit首先描述的测试模式是测试系统时的良好实践。

我们认为，研究界应该效仿其他研究领域的范例，系统地定义、分类、分类和关联模式。我们需要软件模式分类学家来创建模式的统一分类法。此外，这些分类学家应该识别模式语言(如果有的话)，这将帮助开发人员“论述”他们使用模式的系统，从而在模式的使用和对质量的影响方面做出最佳选择。

### Ⅴ 统一原则和特点

我们现在声称，设计模式所提出的设计问题的大多数设计主题都有相同的基本原则：它们引入新的类或方法来添加一个或多个间接级别，并且通过这些间接的级别，提供更大的灵活性。我们现在简要地描述两个例子来支持我们的观点。

访问者设计模式的目的是“呈现对对象结构元素执行的操作”。 Visitor允许你定义一个新的操作，而不需要改变它操作的元素的类。Visitor设计模式提倡的解决方案是引入访问者类的层次结构，这些类符合定义良好的接口，并且由声明到原始对象层次结构中的新方法调用它们的方法。因此，该解决方案建议将执行原始对象中声明的操作的原始方法的一级间接寻址添加到执行在访问者类中声明的操作并由原始对象中的新方法调用的方法。类似地，观

察者设计模式打算通过描述特定类中观察者的行为来定义“对象之间的一对多依赖关系”，这些类的通知方法是由一个主题调用的，而不是直接在主题中实现这种行为。因此，该解决方案还建议从原始主体和观察者的行为中增加一个间接层。

但是，我们也声明，并不是所有的设计模式都基于增加一个间接层以增加灵活性的原则。例如，组合设计模式在设计原则上有一个集合继承关系和组合关系之间的循环。这些例子表明，设计模式的基本原则目前定义不清，尽管它们的定义有两大好处:

1. 根据当前设计模式的基本原则，对其进行系统的形式化和分类，以帮助教学。
2. 系统地结合这些基本原则，从第一原则而不是通过经验来识别新的设计模式。

此外，其他类型的模式也没有明确的底层原则，而这些原则的缺乏阻碍了对这些模式的深入讨论以及对它们对软件质量影响的研究。此外，尽管面向对象的编程语言已经很成熟、流行，并且在实践中已经取得了成功，但是由于缺乏原则，在不同的编程模式（如函数式编程）、不同的设计（如依赖注入设计）中，模式无法相互融合，以及不同的架构风格，例如面向服务的架构。除了形式化、分类和交叉充实之外，研究模式背后的原理可以使模式与抽象层（实现、设计、架构）和范式（面向对象、功能）统一起来。此外，具有统一的基础原则可以帮助强制实施在系统中实现的模式。尽管存在工具来执行模式，例如，Wuyts’ SOUL，但这些工具需要对所执行的模式进行明确的定义。类似地，统一的底层原则将有助于设计生成模式实现的工具——或者重构代码以实现模式。

最后，研究模式的原则也可以为某些实践作为“模式”提供支持或反对的证据。正如Alexander和Gamma等人所建议的，一个问题及其解决方案必须在三种不同的情况下遇到，至少在被命名为“模式”之前。因此，当前一些称为模式的实践并不是这样“真正的”模式。相反，对于常见问题的解决方案，即使在不同的上下文中重复出现，本身也不是“模式”。 例如，使用迭代器遍历列表的编程习惯用法在许多编程语言中都很常见，因此它是语言的一个典型部分，而不是模式。此外，有些模式的存在只是为了克服编程语言中的限制，例如，Java与析构函数中的“使用资源”特性。

### Ⅵ 建议议程

前面几节总结了研究社区在过去几年中在研究和理解设计模式(尤其是不同类型的模式)方面取得的巨大进展。这些部分还突出了社区目前面临的问题。

对所有类型的模式进行了大量优秀的研究，这使得研究人员、学生和实践者很难理解模式之间的差异和共性，也很难评估某些类型模式的某些结果是否听起来不错，但对其他人来说，类似的结果可能不那么好，并从一些类型的对他人的模式。这些不同的实验设计产生了关于模式的有趣的结果，但是很难在实验设计和模式之间进行比较，并且很难理解它们的全球影响。

因此，我们向对模式感兴趣的研究群体提出以下建议。

1. 从开发者衍生的行为模式: 由于开发人员是习惯的产物，在开发和演化(理解)他们的系统时，模式对开发人员最有用。我们建议社区继续研究模式对开发人员的直接影响，例如，在实验设计中使用眼动跟踪器。现在，眼动跟踪器价格低廉，在各个研究领域越来越受欢迎，可以系统地用于研究模式及其对开发人员的影响。
2. 开发者的行为模式：模式可以用作系统的构建块，也可以用作模板，根据模板来评估开发人员编写的代码以及他们在ide中的软件开发活动，并根据模板推荐要遵循的模式。因此，我们建议研究社区开展研究工作，分析通过Mylyn或FeedBaG收集的开发人员日志，并研究模式的使用，以确定低效活动并帮助开发人员。
3. 建筑系统的模式: Rich和Waters[率先将模式(他们研究工作中的习惯用语)作为系统的构建块。很少有作品遵循他们的脚步，这条研究路线似乎大部分都被放弃了。我们建议使用模式提高编程语言的抽象级别，以促进适当的模式作为编程语言的构建块。例如，Oracle在Java 7中引入了自动资源管理的概念，通过try with resources块来实现，从而取代了一个常用的习惯用法。
4. 软件模式理论: 基于现有的工作和可能的未来工作，我们建议研究社区抓住其研究结果所给出的机会，提出一个理论或竞争模式的理论。这样的理论应该解释为什么某些类型的模式会对质量产生积极影响，而另一些则不解释为什么某些类型的模式存在于某些软件构件中。这样的理论可以用来构建实验设计，识别新的模式及其影响。

### Ⅶ 总结

十年前，我们调查了研究社区，了解专家对设计模式的感知及其对不同质量属性的影响：伽玛等人提出的四个属性、灵活性（可扩展性）、模块化、可重用性和可理解性，以及其他六个属性：简单性、可学习性、通用性。运行时的模块性、可伸缩性和健壮性。我们表明，专家认为设计模式并不总是提高质量，一些设计模式对Gamma等人强调的属性有特别负面的影响。

在这项调查之后，其他研究表明，基本上遵循三个(非互斥)研究方向: (1)关于设计模式对质量影响的进一步研究，例如，Zhang和Budgen的文章“关于软件设计模式的有效性，我们知道些什么?” (2)研究其他类型模式的定义和影响，如Di Penta等人的文章《你的代码何时以及为什么》开始有异味(以及异味是否消失)” （3）加强模式出现识别的研究，例如“使用机器学习技术的代码气味严重性分类”。

我们认为这些研究极大地提高了我们对模式的广度和深度的理解。然后，我们建议研究界在这些研究的基础上，进一步研究模式及其对质量的影响，特别是建立模式理论，并探索较少的研究方向，例如，使用模式作为系统的构建块。