经验软件工程的挑战和发展趋势

CCF 软件工程专委会

摘　　要

软件系统在当今世界已经有极其广泛的应用，大量新型的软件相关系统在不断地被开发和使用，譬如社会软件服务系统、软件生态系统等。与此同时，开放和自治的软件开发环境正在彻底变革软件的开发方法、过程。如何处理、理解和利用软件开发中产生的大量数据，使之可以促进新环境下软件质量的管理与改进，是我们面临的严峻挑战。经验软件工程正是这样一门学科，其研究内容涉及如何在软件工程的环境中获得经验数据、如何进行数据分析，从而提出假设并进行科学求证，最终指导软件过程和质量的改进。

关键词：经验软件工程，软件工程，数据分析

Abstract

Nowadays, software systems are being developedand applied in an increasingly wide scope. There are various emerging types of software systems such as Social Software Service Systems and Software Ecosystems. On the other hand, open and autonomy software environments have changed the software development methodologies and processes. How to process, understand, and leverage the data produced in the software development process, facilitate the management and improvement of software quality is quite a challenge. Expirical Software Enigneering is such a discipline, which researches and provides some systematic methods to get data from the environment of software engineering, analyze the data, formulate the hypothesis and evaluate the results. Finally, apply the outcome to improve the software processes and products.

Keywords: Empirical software engineering, software engineering, data analysis

1　引言

任何学科的发展都要依赖于对这个学科所要解决的基本问题的理解。这就涉及对该学科应用领域的建模，以及对问题的解决过程和解决效果的建模。对软件工程而言，不仅要对软件产品的各种特征建模，譬如可靠性、可移植性、效率等，还要对软件项目的各种特征，譬如成本、进度等进行建模。而更重要的是，还要理解开发过程和产品特征之间的关系，例如什么样的开发过程在什么条件下可以产生什么样的产品特征。也就是说，我们不能脱离工程的要素去孤立地看待产品质量问题。

每个学科解决问题的能力都会随着领域经验的提高而提高。其基本方法是把经验封装到模型中，并基于实验、经验证据、反馈来验证和确认模型的正确性。对知识的封装

可以让我们站在更高的抽象层次上理解我们的问题空间、解决空间，并通过应用反馈和学习让我们知道哪些方法行之有效。

这是一种在很多领域都适用的方法，譬如物理、医学、制造业等。以物理为例，它瞄准的是对物理世界的理解，并将研究分为理论研究和实验研究两大阵营，物理学科的进步正是这两大阵营相互影响的结果。理论研究者建立模型来解释世界，预测那些可以度量的事件所可能导致的结果，这些模型既可以基于理论，也可以基于之前的实验数据。实验研究者则通过观察和度量，来证实或者推翻理论，或者探索新的理论。这是一个建模、实验、学习和再建模的闭环。

与上述学科一样，软件工程也需要这样高层次的方法来推进学科知识的发展。软件工程也是一门实验科学，需要建模、实验、协作来理解各种过程和技术工作的状态、制约的因素以及改进的机会。我们也必须从应用中学习并改进自身对软件工程世界的理解。

但软件工程是面向人的学科，与其他工程学科相比，人的智力活动在软件开发中起着决定性作用。人作为软件开发者和工程实践者，通过与软件工程技术进行交互创造出软件系统。因此，在软件工程的研究中，人的因素和影响往往扮演着更重要的角色。没有任何两个软件是相同的。过程是变量，目标是变量，环境也是变量。我们需要建立、分析和评价软件过程、产品以及相关环境的各种有用模型，以及模型之间的交互作用。经验软件工程（Empirical Software Engineering）就是这样一种方法，通过实验手段仔细分析、验证假设以及适用的环境和条件，从而使得在小范围内揭示和确认的方法可以在更大、更复杂的环境中得到应用。

在经验软件工程研究中，研究者和实践者的共生关系相对于其他学科更为明显。研究者的任务是理解开发过程和产品的自然属性以及它们之间的关系；实践者的任务是利用这些知识建立改进系统。研究者需要的实验室中通常有实践者建立的软件系统，而实践者则需要研究者提供的模型去建立更好的系统。

经验软件工程的起源可以追溯到 20 世纪 70 年代 Weinberg 和 Ticky 的工作。他们提出并鼓励采用科学的方法来细致地观察和评鉴软件开发方法。最早的实验出现在 20 世纪 80 年代早期，Victor Basili 在 Maryland 大学和 NASA 的软件工程实验室的团队，使用实验的方法来检查软件工程技术的适用性。

进入 21 世纪后，随着互联网技术的发展，我们可以看到软件的规模、复杂度在戏剧化地攀升，系统中的系统（System- of- Systems）、社会软件服务系统（Social- Software-Service-System）、软件生态环境（Software- Ecosystem）等新的软件形态和社区正在形成。2013 年 9 月，Gartner 在对 2014 年具有战略意义的十大技术与趋势做出总结的时候，首次引入了“软件定义一切（Software Defined Anything）”的概念。事实上，软件定义数据中心、软件定义网络、软件定义系统等正在成为业界共识，软件的地位变得越来越突出。这种发展也意味着承载其服务的软件的需求爆炸性增长，传统以公司为主体的软件开发模式已经无法满足超量的软件需求，市场呼唤一种快速、低廉并且高质量的软件开发方法——开源/开放的软件开发环境。开放环境的协同模式改变和颠覆了软件业的工作方式，真正释放了软件开发人员的创造力与生产力。从业者通过自愿、兴趣和协作的方式，

依托互联网的交互平台形成软件的开发和贡献群体。自由、开放地交流沟通、共享经验、参与协作，给软件的开发活动融入了更多的社会学元素，也带来了更多的不确定因素，使得软件工程这个社会－技术系统变得更加复杂。其突出表现在以下几个方面。

1）**复杂社会－技术系统中的大数据**。传统软件开发环境中已然存在大量的数据，譬如各类工程文档和过程管理数据，一直存在的问题都是数据多、质量低，缺少使用的方法。因此很多的数据都是束之高阁，无人也无法使用。在开放环境下，预定义、有结构的开发方式被逐渐摒弃，文档少了、评论多了，数据的噪声也更大了。人们在畅快淋漓地写代码的同时，发现软件的维护和演化变得如此困难，到底该做什么？怎样才能给我们带来最大的价值？如何评估风险？究竟该推荐什么人才能高质量地完成任务并保持社区的繁荣？没有确定的过程可以遵循，没有明确的资源可以使用，没有合同约束你该做什么，这就是开放环境带给我们的挑战。经验软件工程的核心思想就是在观察和推理的基础上探索理论。开放环境给软件工程带来了挑战，同时也带来了机遇。观察开发过程中的数据、分析数据间的关系，进而推理出合适的理论，这一过程有助于我们在软件开发活动中开展评估、选择和决策工作。

2）**软件开发的效率和质量**。开放环境中的软件涉众从封闭、有限的群体转变成开放、无限的群体，社区内的成员可以自由、开放地交流沟通、共享经验、参与协作，软件的开发活动融入了更多的社会学元素。虽然利用大众的群体智慧为快速、低廉地开发高质量软件提供了可能，但生产方式的变革要求我们必须重视社会学形态对软件开发的方式、过程、生产率和质量的影响。譬如开源软件由于其松散的组织方式、自由的开发贡献和无承诺的交付方式，使得大量的开源软件存有各种缺陷和安全隐患。而这些问题又由于生产方式的变革，使得传统的方法难以发挥作用。因此，封闭环境中的效率和质量的提高依赖于过程和方法的改进，而在开放环境下，由于需求、资源和边界的不确定，这要求我们不得不考虑更加柔性、多变的改进方法。譬如准确的资源推荐而不是分配，基于大众使用评论的质量评价和改进等，这也给经验软件工程带来了更多的应用和发展空间。

3）**开放社区中的群体智慧**。开放社区建立了一种自治合作的软件生态环境，不仅是软件的开发者，软件的用户也以各种方式参与其中，形成了一种多层次、多元化的软件涉众群体。在这种自治的软件生态环境中，社会学的属性会映射到个人和群体的行为中，传统的软件过程方法已经不适应这种社会自治的合作方式，大众在竞争与协作中形成的各种知识必然会以某种方式促生新的群体智慧。如何理解开放环境下群体智慧的复杂性，如何建立支持开放、自治的社区合作软件开发模式，如何适应群体合作、演化和发展的软件过程方法，以及基于群体智慧的需求分析、代码理解、缺陷预测/定位/修复等如何支持软件质优价廉的开发，这些用经验软件工程的方法都可以找到合适的解决方案。

因此，我们看到，经验软件工程从最初的软件工厂开始发展，今天在软件生产方式上有重大变革的时机，其基于观察、实验和理论推理的方法对不确定的软件开发环境中新问题的分析、理解和改进提供了有效的解决途径，特别是针对软件工程中的大数据与基于大数据的知识获取、分享、运用，已经有了非常重要和前瞻性的研究，并且将推动软件工程领域的技术发展。

2 经验软件工程的基本研究方法

Empirical 在英文中是“经验的”或“实证的”意思。因此，Empirical Software Engineering 被译为“经验软件工程”或“实证软件工程”。但在英文中 Empirical 区别于 Experiental，这里的“经验”不单单是人在实践中的主观体验和认知，更强调从实践中获得的客观证据。

经验软件工程的研究内容涉及如何获得经验数据、如何进行数据分析，从而提出假设并进行科学求证，最终指导实践活动。根据数据形式的不同，经验研究又可分为定性研究（Qualitative Research）和定量研究（Quantitative Research）。定性研究中处理的数据主要为文字、图像等难以量化的信息，其研究方法主要来自社会科学领域，如编码（Coding）、主题归纳（Thematic Synthesis）、持续比较（Constant Comparison）、扎根理论（Grounded Theory）等。定量研究中处理的是能够被量化的数据，会用到概率统计（如假设检验、回归建模）、仿真试验、机器学习等方法。通常根据数据获取的方式、研究对象、分析方法、证据强度等的不同将这些研究方法分为受控实验（Controlled Experiment）、调研（Survey Research）、案例研究（Case Study）和行动研究（Action Research）等，这些方法虽然各有特点，但并不排斥，也常常混合使用。比如通过案例研究发现可能的因果关系，再通过更广泛的调研或设计受控实验进一步求证。在关于相似问题的大量第一手经验研究的基础上，还可以使用系统评价（Systematic Review）开展二级经验研究。

经验研究是以改进软件工程实践为目的，将理论上的概念运用于软件工程实践中，还将应用经验形成知识加入软件工程系统中，以改善过程、方法和工具，验证理论与模型。在软件工程中，我们需要对理论和“一般常识”进行经验验证，判断是否存在某种关系（如可维护性及其度量因子之间的关联）需要选择模型、技术或工具，并判断它们在实践中的效果。

经验软件工程的研究目标是实施更好的经验研究、注重因果机制并生成理论，实现迭代和改进。用良好的经验能够更迅速地构建软件工程知识体系，快速删除低回报的想法，识别和评估高回报的想法，探究重要的实际想法。经验软件工程将受控实验、调研、案例研究等研究方法应用到软件工程研究中，积累软件工程领域的经验知识。

下面我们就软件工程中的主要经验研究方法和需要考虑的问题做一些简介。

2.1 受控实验

受控实验是一种实验性研究方法，它针对一个或多个可验证的假设，并对其中相互独立的自变量（Independent Variable）加以控制，系统地操控实验条件，尽量消除非研究自变量的影响，并观察和验证被研究的自变量是如何影响某些可度量的结果的，即因变

量（Dependent Variable）。

大部分受控实验的主要目的是验证某个假设或者关系。假设检验一般是针对前者，后者主要是基于所收集的数据建立一个关系模型（如预测模型），这个模型通常可以用多元统计方法得到，例如回归技术，然后用实验进行评价。

实施一个实验涉及几个不同的步骤。这些步骤是：确定范围、计划（包括实验人员、实验环境、实验假设的阐述，确定实验变量，实验方法和过程的设计等）、操作（包括实验准备、实验执行和数据验证等）、分析和解释、展示和打包（如论文发表等）。

2.2　调研

调研的目的是为了给出与特定研究问题相关的观点的全面总结，一般通过收集来自于人或者与人有关的信息，来描述、比较或者解释人们的知识、态度和行为。在实际应用中，基于调研的方法有 3 种形式，即访谈、问卷、文献。这几种方法并不是互斥的，很多研究人员将几种方法混合使用。通过数据三角检验法往往能设计出更合理的调查，甚至得出更有说服力的研究结果。

Kitchenham 等在 2002 年发表了一系列文章[219]~[221]，系统介绍了调研方法的原则，这些原则也成为软件工程领域进行调研研究的执行指南。此后，调研方法在经验软件工程领域越来越多地被采用，其优点是能够直接、全面地获知受访对象的观点、经验等第一手数据。然而由于这一方法受人为因素影响较大，往往难以有效地避免访问者与被访者的个人偏见，同时由于采样方法多基于便利条件，很难保证调查结果的有效性。这些都是软件工程研究人员使用调研方法时需要考虑和关注的问题。

基于文献的调研在最近 10 年得到了快速的发展。2004 年，Kitchenham[219] 提议软件工程研究者应当开展基于证据的软件工程（Evidence-based Software Engineering，EBSE），并为此正式在医学领域引入了系统文献综述方法。这是一种是针对某个特定的研究问题、主题领域或现象，识别、评估及解释与之相关的所有可用研究的手段。相对于原始研究（Primary study），系统文献综述是一种二次研究（Secondary Study）。

传统综述更多的是基于专家的经验和判断，强调权威性。系统文献综述则更强调客观性，即强调分析和结论的客观性，因此要求对调研过程要有详细的描述。系统文献综述有以下特点：

- 系统文献综述要求在开始时定义评价协议，在协议中声明文章的研究问题以及分析将使用的方法；
- 系统文献综述是基于一个清晰定义的目的而尽是完整地获取相关研究的搜索策略；
- 系统文献综述会记录所使用的搜索策略，以便于读者能够认知它的准确程度和完整程度；
- 系统文献综述需要明确的收录和排除标准来评估每一篇可能的原始研究；

- 系统文献综述会列出即将从每篇原始研究中获取的信息（包括质量标准），以便于评估每篇原始研究。

2.3 案例研究

案例研究是“以典型案例为素材，并通过具体分析、解剖，对当前某一现实环境中的现象进行考察的经验性研究方法”。案例研究方法非常适用于各种软件工程研究，因为软件开发过程是一个极其复杂的过程，往往很难孤立地研究各个因素；同时基于受控实验的成本等因素，它往往难以模拟大型复杂系统，也难以考虑政策等外界因素，因而案例研究室是对受控实验的一个很好补充，它包含了很多实验无法体现的特质，比如规模、复杂度、不可预测性以及动态性。

对于多种类型的软件工程研究，只要被研究对象是当前存在的且难以单独研究的，都可以采用案例研究的方法。案例研究不会产生和受控实验一样的结果，如因果关系等，但它可以让我们对真实环境中的现象有更深入的理解。正因为与分析型、受控的经验研究不同，案例研究总被批评为价值小、无法泛化、带有研究者偏颇性等。但如果在实践中能采用合适的研究方法，并接受“知识不仅是统计学意义的”观点，则上述批判就不是问题了。

案例研究也非常适合对软件工程方法和工具在工业界的使用情况进行评估，因为它可以避免按比例放大问题。但在进行案例研究时，必须设法最小化混杂因子所带来的影响。

在软件工程中，经验研究以及它们的影响在不断增大。然而经验研究方法论大部分集中在实验研究中，因此以分析为主的研究范式不足以反映复杂的现实生活问题，尤其是涉及人类以及他们的互动技术方面的问题，为应对复杂问题，通常采用案例研究。Runeson 和 Host 给出了软件工程学科的案例研究的框架，这篇论文具有里程碑意义，之后的大部分相关论文都是根据这个框架做案例研究的。

案例研究的步骤包括：①案例研究设计（定义目标和制定计划）；②准备数据采集（定义数据采集的程序和协议）；③收集证据（介绍数据采集的过程，并收集数据）；④对数据进行分析处理并讨论；⑤撰写案例研究报告。这个过程和其他类型的经验研究几乎是相同的，然而，案例研究的方法具有灵活的设计策略，那么它在步骤上会产生大量的迭代。

案例研究是对现实世界问题进行的研究，其目的是在控制水平和现实程度之间寻找一种平衡。现实的情况往往是复杂和不确定的，因而案例研究具有高度的现实主义。案例研究的优势在于它更容易设计并且更加真实，劣势则体现在它的结果难以归纳，也更难解释。

2.4 经验数据的处理和分析方法

在经验研究中，需要通过对调研、案例及实验中的数据做出分析，从而得到一些结

论。经验研究中常见的对数据处理的方法包含：数据预处理、数据统计分析、假设检验及数据分类和预测。

数据预处理是为了提高数据处理的质量，预先对数据中的虚假数据点等进行过滤处理，以提高后续分析的质量和效果。数据预处理的方法有多种：数据清理、数据集成、数据变换、数据规约等。数据清理通过填写缺少的值、光滑噪声数据、识别或删除离群点并解决不一致性来“清理”数据，从而达到数据格式标准化、异常数据清除、错误纠正及重复数据清除的目的。数据集成则是将多个数据源中的数据结合起来并统一存储。数据变换则通过平滑聚集、数据概化、规范化等方式将数据进行转化。数据规约可以通过规约表将数据的量减少但仍能基本保证数据的完整性，其结果与规约前的结果相同或几乎相同。

数据统计分析是指运用统计方法及分析有关的知识，结合定量与定性，进行研究活动。数据统计分析需要描述所分析的数据性质、研究群体的数据关系，创建模型，总结数据间的关系。其目的是分析出差异、趋势，查找问题。常用统计分析包含对数据集中趋势、离散程度、数据间依赖关系的度量，以及对数据的图形可视化。

假设检验是数理统计学中根据一定假设条件由样本推断总体的一种方法。假设检验的目的是确定是否能够基于某统计分布的一个样本拒绝某个空假设 H_0。检验可以分为参数检验和非参数检验。常用的假设检验方法有：t－检验（t-test）、曼－惠特尼检验（Mann-Whitney）、F－检验（F-test）、配对 t－检验（Paired t-test）、威尔科克森检验（Wilcoxon）、符号检验（Sign test）、方差分析检验（ANOVA）、克鲁斯卡尔－沃利斯检验（Kruskal-Wallis）、卡方检验（Chi-square）等。

数据分类和预测则是从以往的数据中获取数据分类，从而预测、应对未知的东西，主要是使用机器学习来实现分类和预测。分类预测的算法根据数据功能和形式的类似性开展工作，常用的算法有：决策树学习法、贝叶斯方法、基于核的算法、人工神经网络、集成算法等。

2.5　经验分析的质量与实验有效性验证

除了上面介绍的经验研究的主要方法和技术，在软件工程中开展经验研究还涉及一些相关的重要问题，这在几乎所有的经验研究中都要考虑到。

2.5.1　数据质量分析

软件开发数据的快速积累为软件工程研究者提供了巨大机遇，使软件工程研究者得以基于大量数据对研究问题进行定量研究。与此同时，越来越多的机器学习、数据挖掘算法被引入软件工程研究中，以帮助软件工程研究者更好地理解、利用软件开发数据。然而，软件开发数据自身的质量却没有被很好理解。这些数据可能由于项目的实践不同、工具使用者的使用方式不同，并不能忠实地反映软件开发的实际过程或并不符合软件工

程研究者对其所做的假设。例如一个项目在其演变过程中会更换版本控制系统，造成数据不完整等问题。若软件开发数据的质量存在问题，由这些数据所得到的结论的可信度将会受到影响。

目前，软件开发数据中的质量问题受到了越来越多研究者的关注。首先，对数据分析方法的研究是最近的一个趋势，其次，探讨数据本身存在的问题及其对研究和实践的影响也正在逐步成为一个重要方向。

2.5.2 经验证据的强度

经验软件工程中的一个重要问题是：在软件工程领域的原始研究的质量通常不高而使用的研究方法各异的情况下，我们该如何对已有的经验研究进行恰当的评价和比较。为此，证据的强度便成了一个很重要的属性。

目前存在着一些能对证据的强度做出判断的系统，其中大部分都认为随机性的实验属于高等级，而观察性的研究（如案例研究）和研究者的个人观点（如调研）则属于相对低等级。不过这也存在着一个固有的缺陷：随机性的实验并不是在所有的研究中都可行，而在某些情况下，观察型的研究反而能够提供更好的证据。

通常来说，以下因素会降低证据的强度：研究质量的限制、结果的不一致、相关程度的不可靠性、不严密或者数据稀少、报告中的偏差。而可以增加证据强度的因素有：联系的强或者非常强的证据、应梯度（Dose- Response Gradient）的出现、对影响的低估等。

2.5.3 经验研究的有效性问题

在经验软件工程中，研究者首先要了解的问题就是经验研究结果的有效性（Validity），以及影响有效性的不利因素，即有效性威胁有哪些。有效性具有推断的性质，每个经验研究结果的有效性都面临着一定范围的威胁。如同风险管理，研究者需要知道威胁有效性的相关因素有哪些，以便适当地控制或者接受这些威胁。为了帮助识别和处理这些威胁，Wohlin 等人把经验软件工程中的有效性威胁主要分为 4 种类型：结论有效性、内部有效性、构造有效性和外部有效性。

1）结论有效性（Conclusion Validity）：如何确保我们在一个经验研究中使用的方法、策略确实与我们观察到的实际结果有关。也就是说，这种结果是否具有统计意义。

2）内部有效性（Internal Validity）：我们如何应用这种方法得到需要的结果，当然，也可能有一些不受控制的因素会导致实际的结果。

3）结构有效性（Construct Validity）：经验研究背后的理论与观察结果之间的关系。即使我们已经建立了实验方法和观察结果之间的因果关系，其方法也有可能不受我们控制，以至于结果偏差很大。

4）外部有效性（External Validity）：我们是否可以将结果的概括总结应用到其他的研究领域。因为即使已经确定我们的研究结果具有统计意义，也不能轻易下结论说这种

方法具有普适性。

2.5.4　经验研究的复制

复制（Replication）是一种经验软件工程研究中非常常见的研究形式。复制研究旨在帮助研究者建立、完善某个特定研究主题中结果和相关上下文的系列知识［Basili99］。在经验软件工程的研究（特别是实验）当中，实验的上下文环境（Context）往往会对实验的结果产生很大的影响，因而，考察某个软件工程实践、方法以及工具在不同的上下文中是否仍然有效就成为经验软件工程研究的主要目标之一。复制是实现该目标的有效手段。通过复制研究，不管发现的结果与原始研究是否一致，复制研究都被认为是对原始研究结果的有效补充。Shull 等人将复制研究分成两种，一种是严格复制（Exact Replication），另外一种是概念复制（Conceptual Replication）［Shull08］。前者要求尽量保持复制研究中的上下文与原始研究中的上下文一致或者接近，后者则仅仅只是针对相同的研究问题，而上下文环境，甚至实验过程都可以不同。

3　软件工程各领域的主要经验研究成果

我们在软件工程学科当中选择了经验研究较多的几个领域，对经验研究在这些领域中的主要有代表性的成果进行总结归纳。

3.1　估算与预测

根据文献［101］的总结，软件成本估算相关的研究最早可以追溯到 20 世纪 60 年代。经过近 50 年的发展，软件成本估算技术取得了长足的发展，但成本估算仍然是目前软件项目管理中的主要难题之一。在软件工程实践中，“软件成本”是一个相对模糊且宽泛的概念。根据应用场景的不同，软件成本估算的目标可能涵盖货币成本、工作量、资源、进度等多个方面。考虑到软件开发的高智力特性，人员成本一般占整个软件项目成本的绝大多数，因此在很多情况下可以用“工作量”（Effort）来代替“成本”的概念。在软件成本估算的相关研究中，大多也将“工作量估算”视为成本估算的目标。

预测是经验软件工程中另一个重要的研究领域，预测主要是研究未来的一种倾向或者趋势，譬如某些模块是否更容易出现缺陷，某些人修复缺陷的质量更高等。

在经验软件工程中，估算和预测研究的主要方面有以下几类。

3.1.1　成本估算

软件成本估算研究发展至今，已有大量估算方法、模型被提出和使用。根据使用的估算方法不同，软件成本估算主要可分为专家估算、基于类比的估算、基于参数化模型的估算、基于机器学习技术的估算和组合估算方法等[87,96,126,23,26,190,193,82-82]。

近年来，针对软件成本估算领域，很多研究人员针对软件成本估算的不同方面给出了系统评价。

2005 年，为了确定在软件工程成本估算中经验研究结果的一致性，文献［114］利用预先建立的搜索准则，构建了一个详尽的文献检索，研究结果表明 25% 的研究结论是不确定的。相关研究者建议在以后的研究中需要关注更细节的问题。

2007 年，文献［101］中全面回顾、分析了软件成本估算的各种代表性方法，也归纳、讨论了与成本估算强相关的软件规模度量问题，进一步研究了软件成本估算方法的评价标准，并给出了一个应用实例及其分析，最后从估算模型、估算演进、估算应用、估算内容、工具支持和人力因素 6 个方面，指出了软件成本估算方法在下一步的主要发展趋势。

在使用 COCOMO II 等参数化模型时，常需要使用本地数据对模型参数进行校准。局部校准被认为是使用参数化模型的最佳实践之一[132]。然而，由于本地数据的积累要耗费很长的时间，消耗大量的数据收集和维护成本，所以在很多情况下需要借助其他组织的项目数据进行成本估算，即进行跨组织估算。目前在跨组织估算和同组织估算的性能比较方面存在争议，尚无统一定论。Kitchenham 等人的对比研究表明，同组织估算的性能显著高于跨组织估算[89-90]。Jeffery 等在 ISBSG 数据集上进行的实验结果也支持这一结论[79]]。类似的研究还有 Mendes 等使用 web 开发项目的数据进行的对比实验[121]。与之相反，Wieczorek 和 Ruhe 等发现跨组织估算的性能与同组织估算的性能无显著差异[195]。Mendes 等人的另一项实验也表明在部分数据集上使用多组织数据和同组织数据建立的成本估算模型的性能差异不明显[122]。

另有部分研究关注如何提高成本估算数据的可用性。如 Kitchenham 通过分析数据集的不均衡性，帮助解释 COCOMO II 模型在最初校准时的困难[88]。Liu 等提出了一个针对软件工作量估算数据的预处理框架，用于识别和剔除异常数据点[106]。

3.1.2 工作量估算

工作量估算是软件项目开发的重要环节，也是制定项目开发计划的依据。现有很多经验研究是针对工作量估算的。

2005 年，文献［167］关注于项目早期的特征子集选取和工作量估算。相关研究者提出了一个使用灰色系统理论中的灰色关联分析的新颖方法，以解决小样本不确定性。他们采用灰色关联分析进行特征子集的选取和工作量的估算。实验结果显示该方法优于其他机器学习方法，有很大的潜力。

2006 年，文献［168］使用基于关联规则挖掘的方法预测缺陷的关联性和缺陷修正工作量。他们把该方法应用在超过 15 年的 200 个项目中，结果表明对于缺陷关联性预测，它的精确度很高，假阴性比率很低；对于缺陷修正工作量，缺陷隔离工作量预测和缺陷修正工作量预测的准确率都很高，并且和别的方法相比，准确度提高了至少 23%。

2007 年，文献［113］调研了 1995 ~ 2005 年间工作量相关的研究，从中抽取了 10 篇相关的研究。分析结果表明由于不同研究使用了不同的数据集和不同的实验设计，导致

了结果的异质性。

2009 年，文献［191］提出了一种基于灰色模型 GM（1，1）和 Verhulst 的方法去预测软件物理时间的阶段工作量。在大型的现实世界的软件工程数据集上验证该方法，并且将其与线性回归方法和卡尔曼滤波方法相比，结果显示它的精确度至少分别提高了 28% 和 50%，从而表明了该方法具有一定的潜力。

2010 年，文献［67］调研了中国工业界中影响开发工作量的因素。他们在 140 个软件组织的 999 个项目中验证了 6 个因素（项目规模、团队规模、工期、开发类型、业务范围、编程语言）对开发工作量的影响。结果表明这 6 个因素对开发工作都有影响。结合这些影响可以更好地对项目进行管理。

3.1.3　规模估算

目前主要的软件规模度量方式有代码行、功能点、对象点和用例点 4 种[101]。以下分别对基于这 4 种度量的软件规模估算的经验研究进行介绍。

1）代码行（Lines of Code，LOC）是一种对软件规模最简单最直接的一种度量，在软件成本估算中被大量使用[4,138,143]。根据度量对象的不同，代码行度量可分为逻辑代码行度量和物理代码行度量两种[95]。目前在软件成本估算中使用较多的是逻辑代码行。概念数据模型在信息系统的需求分析中被广泛应用，信息系统中的很多特征都可以在概念数据模型中体现，文献［178，179］使用概念数据模型来估算代码行，并且在开源系统和工业界验证了该方法的有效性。

2）不同于代码行度量，软件功能点（Function Point）度量方法主要基于用户需求对软件规模进行估算，估算结果独立于编程语言类型[4]。目前应用较为广泛的一种功能点计算方法是国际功能点用户协会提出的 IFPUG 方法[73]。其他的功能点度量方法还包括 Mk II FPA 方法[187]、NESMA 方法[133]和 COSMIC-FFP 方法[188]等。功能点主要基于软件的功能需求来估算产品的规模，缺乏对软件系统复杂度的考虑。此外，功能点方法的适用范围有限，一般认为其比较适合管理信息系统，而对于逻辑复杂的科学计算软件等不太适用。

3）对象点（Object Point）方法与功能点方法的原理相似，通过统计需要耗费大量工作量的制品来估算软件规模，如服务器数据表的数目、客户数据表的数目、报表和屏幕视图中的可重用比例等[13]。

4）用例点（Use Case Point）方法是随着统一建模语言（Unified Modeling Language，UML）在软件开发中逐渐普及而兴起的另一种软件规模度量方法，目前在成本估算中也得到了广泛的应用。用例点方法的典型代表是 Karner 于 1993 年提出的 UCP 方法[85]。

3.1.4　软件维护预测

为了增加新功能、修复缺陷或者适应新环境，软件需要不断地更改，从而会变得越来越复杂，维护成本越来越高。因此，对软件维护有效方法的研究引起了越来越多研究人员的关注，如预测维护成本[100,211]，评估可更改性[32]，预测软件中可能发生更改的部分[8]等。

更改预测可以使软件开发者和维护者更有效地利用诸如同行评审、测试和检验等资源预测面向对象软件系统中哪些模块将可能发生变更，这在软件工程领域具有重要意义，它们可以作为工作量调度依据，有助于提高软件开发和维护工作的效率。到目前为止，研究人员提出了许多通过软件历史数据信息和当前源代码信息来预测模块变更趋势的方法。在预测更改的可能性的多种方法中，使用软件度量构造更改可能性预测模型是一种广泛使用的方法[93,65]。静态软件度量不需要运行软件就可以直接得到，因此引起了很多研究人员的关注[100,211]。Chaumun 等[32]提出一种变更影响模型分析在软件系统各个模块中的传播以及相互影响；Zimmermann 等[216]提出利用版本控制系统中的数据来预测软件系统在未来的版本变更趋势；文献［185］提出了用一个概率方法去估算当一个功能增加或修改时更易改变的类。Sharafat 等[160]提出一种基于模块依赖和软件历史数据信息的模块变更概率预测方法；薛朝栋等对 Sharafat 等[160]的变更预测方法进行简化，在软件依赖关系的基础上提出了一种轻量级的模块变更概率计算方法。文献［109］采用统计元分析方法在 102 个 Java 系统中去探究 62 个面向对象度量元预测易出缺陷类的能力。文献［214］搜集了类级别上的静态代码度量元和变化数据，分析发现 80% 的代码行变化存在于 20% 的类中，然后他们使用分类技术来预测易改变的类，结果显示他们提出的分类方法对于易改变类的预测是有用的。

3.1.5 缺陷预测

作为对软件质量保障具有重要实际意义的一项活动，软件缺陷预测从 20 世纪 70 年代即开始受到关注，至今仍然是软件工程领域的一个热点研究问题[1,199,80]。目前，学术界普遍存在的一种公认的观点，即由于软件项目的复杂性，在不能确保软件项目不产生缺陷的前提下，将关注点转向在软件发布早期对软件缺陷进行合理的预测，尽量减少经发布的软件中可能存在的缺陷，并且，在软件发布之后对用户报告的缺陷及时有效地解决，以减小由于软件缺陷给用户带来的负面体验。基于此种观点，研究人员提出了一系列方法以支持软件发布之前的缺陷早期发现和软件发布之后的有效解决，主要成果体现为以下 4 个方面：

1）**缺陷预测**。这方面的研究将数据挖掘、机器学习等方法与软件产品、人员和过程度量方法相结合，利用大规模的历史软件开发和缺陷数据，通过学习已知模块或组件的度量特征与缺陷之间的潜在模式，预测新增模块或组件中可能存在的缺陷。例如，文献［210］利用面向对象设计度量元预测高、低严重程度的缺陷。Zimmermann 和 Nagappan[217]利用代码片段之间的依赖关系构建社会网络并利用社会网络度量元进行代码缺陷预测。Kim 等[91]研究了缺陷数据中的误分类数据对缺陷预测模型的性能影响。Li 和 Zhang 等[102]将主动学习、组合学习和半监督学习的思想引入软件缺陷预测中，针对可利用的软件缺陷历史数据稀少珍贵的问题，提出了基于抽样的主动式半监督缺陷预测方法。2014 年 Yuan 等[205]人提出了面向细粒度源代码变更的缺陷预测方法，他们采用特征熵差值矩阵分析了软件演化过程中概念漂移问题的特点，并提出一种伴随概念回顾的动态窗口学习机制来实现长时间的稳定预测。

经过几十年的研究，软件缺陷预测问题取得了长足的进展，有大量的预测方法被提出和验证。然而，有研究表明，已有的缺陷预测方法大多仅在项目内适用，即需要采用同版本或同项目历史版本的缺陷数据来进行预测模型的训练[218]。产生这一限制的原因是传统的机器学习技术是基于数据同分布假设的，这要求预测模型的训练数据与测试数据（或待预测数据）分布一致。在某些实际的软件缺陷预测应用环境中，这种同分布的本地历史数据不可获取。为解决本地历史数据缺乏的问题，一个可能的途径是进行跨项目缺陷预测（Cross-project defect prediction），即参考外部项目的历史数据（跨项目历史数据）中的缺陷预测模型，然后将其应用到当前待预测项目上。跨项目缺陷预测问题近年来得到了大量的关注，有部分研究对跨项目缺陷预测的可行性、预测算法、数据预处理方法等进行了初步探索，并取得了不错的成绩[111,141,227]。

近年来很多研究者对缺陷预测研究做了系统评价，比如2013年，文献［147］的相关作者调研了1991～2011年间106篇与缺陷预测度量元相关的文章。结果表明面向对象的度量元的使用几乎是代码度量元的两倍，其中CK度量元是最常用的；面向对象和过程度量元比规模和复杂性度量元的缺陷预测能力更强，其中过程度量元比任何静态代码度量元更能较好地预测发布后版本的缺陷。同年，文献［74］分析了哪些面向对象度量元对预测易出缺陷模块是有用的，相关研究者调研了已有的基于CK＋SLOC度量元的经验研究，并从中提取出29篇相关文章。研究结果表明，耦合性、复杂性和规模度量元对易出缺陷模块的预测有更强的影响。2014年，文献［76］分析了面向对象度量元与外部质量属性（可靠性、可维护性、有效性和功能性）之间的关联性，相关研究者调研了99篇相关文章，结果表明与继承度量元相比，复杂性、内聚性、规模和耦合性度量元与可靠性、可维护性有较强的关联性。2015年，文献［115］等相关作者调研了1991～2013年间使用机器学习方法的缺陷预测论文，并从中筛选了64篇文献，同时把机器学习方法分为7类。通过分析，总结出易出缺陷模块预测中机器学习技术的使用情况，评价了其预测准确度和能力，并与统计学方法进行了对比，最后总结出机器学习方法的强项和弱项。

2）**缺陷诊断**。这方面的研究主要关注于在软件缺陷发生之后，如何根据提交的缺陷报告将软件缺陷及时有效地分配给相关开发人员，以进行缺陷修复。因为开源软件项目缺乏统一的开发任务管理，并且由于个人开发人员对开源项目的开发兴趣具有不确定性[207]，致使个人开发人员不可能了解所有软件组件的依赖关系和各组件的具体代码实现。在该情况下，如何在有限信息和不确定条件下进行软件缺陷诊断已经成为一个越来越突出的问题。例如，Anvik等[9]利用文本分类方法进行缺陷修复人推荐。Xie等[200]等利用LDA主题模型针对不同开发人员对于修复缺陷的不同程度兴趣和专长进行建模，并在此基础上提出DRETOM方法以进行缺陷修复人推荐。文献［39］提出一种基于调用堆栈重复缺陷报告聚类的方法。文献［65］分析了一个缺陷报告是真正的缺陷还是一个过时的测试。文献［209］预测缺陷可能在多长时间内修复。

3）**缺陷定位**。这方面的研究主要关注于在开发人员接受缺陷修复任务之后，如何依据缺陷报告描述内容、代码变更历史、邮件交流、调试运行信息、软件功能演化、运行

日志记录和个人开发经验等，理解缺陷发生的原因，定位可能导致缺陷发生的程序代码片段。尤其是在开源软件开发由开发人员意愿驱动，缺乏明确且严格描述的需求文档和测试用例的条件下，使得一些自动化的缺陷定位和修复方法不具可行性。因此，开源软件在缺陷发生之后，很大程度上要依赖于开发人员的个人能力来理解、调试和变更代码以进行缺陷修复。于是，如何提供有效手段来辅助开发人员快速定位导致缺陷的代码、理解缺陷发生的原因并提供可能的缺陷解决方案备受是软件工程研究人员关注的另一重要问题。例如，Stacy 等[169]基于 LDA（Latent Dirichlet Allocation）方法建立源代码主题模型，将源代码文件用潜在语义主题进行表示，并从缺陷报告中抽取出用户查询以定位修复缺陷时所需修改的源代码文件。Moin 和 Khansari[127]提出了一种基于代码修改日志和缺陷报告文本分类的方法，以定位程序源代码目录级别的缺陷。Zhou 等[213]参考缺陷报告的文本长度和缺陷报告之间的文本相似性提出了一种基于信息检索的缺陷定位方法 BugLocator。文献［186］利用方法调用异常来提高基于频谱的缺陷定位技术在面向对象程序中的有效性。文献［213］利用信息挖掘技术预测缺陷的修复位置。文献［58］提出一种利用简单的用户反馈的交互式缺陷定位方法。文献［124］提出了一种基于聚类的策略来确定故障定位的准确性。文献［197］利用分段和堆栈跟踪分析来提高面向缺陷报告的缺陷定位的准确度。

4）**评审后缺陷预测**。评审后缺陷预测是指，采用一定的技术和方法预测未被评审发现的缺陷。该课题的相关经验软件工程研究可以大致分为两个方向：①利用历史数据建模进行预测，其中又可以细分为跨项目（Cross-project）模型和项目内（Within-project）模型；②不依赖历史数据进行预测，目前主要采用的是在生物统计学中已经非常成熟的捕获再捕获（Capture-Recapture，CRC）模型。目前，该领域相关研究中绝大部分都是基于历史数据的缺陷预测技术。

基于历史数据的预测模型往往会受限于历史数据的质量和数量，因而，建立起来的预测模型有一定的局限性。例如，由于大多数的新项目并没有历史数据，所以无法使用项目内模型；而 Zimmermann 等人的研究表明，即使是同一个领域的项目，采用跨项目模型来预测缺陷也会得到糟糕的结果[218]。

如果不依赖历史数据，无疑可以大大增加预测技术在实际软件项目评审中的应用可能。其中，捕获再捕获模型是一个值得关注的方向。然而，Liu 等人采用系统评价的研究方法综合分析了目前 CRC 模型的相关研究。其结果表明，在基于 4 个基本 CRC 模型的 18 个具体的计算算子中，并没有一个相对确定的最好算子，并且，缺乏工业领域的应用也是 CRC 模型相关经验研究当前状态存在的问题之一[107]。

除了对于缺陷总数的预测，还有一些关于缺陷位置的预测研究。如，Rahman 等人比较了 SBF（Static Bug-Finding）和 SDP（Statistical Defect Prediction）。前者是一种在编写代码的时候提示可能的缺陷及位置的工具，后者则通过建模预测代码在哪些地方可能存在缺陷。研究表明，两者的成本收益比相近，并且 SDP 可以提高 SBF 的表现[148]。

3.1.6 资源估算

除了上述几种估算，还有资源估算[11]。网络的发展在当今的产业中占有重要地位，

文献［11］深入研究了网络资源的估计，并给出了一个网络资源估计的系统评价，以确定当前网络资源估计的发展现状与可能存在的研究间的差距。他们首先确定当前的研究现状，在执行一个全面的文献检索后，选取了84篇经验研究，以便于调查网络资源估算的各个方面。研究结果表明，现今没有一个在特定估算场景下该使用哪些资源估算技术、怎样执行、怎么评价其效率的指导；精确度的结果也有很大的不同，并且依赖于许多因素。重心在开发工作量/成本估算的研究上，忽略了其他方面的资源估计，如质量和维护；规模的度量已经应用在很多地方，但是只在一个研究中被作为一个资源的指示器。他们的研究结果表明，在网络资源估算领域还有大量的工作可做。

3.1.7　国内研究现状

成本估算方面。西安交通大学的宋擒豹等[167]关注于项目早期的特征子集选取和工作量估算，他们提出了一个使用灰色系统理论中的灰色关联分析的新颖方法，能够解决小样本的不确定性。接着，他们[168]使用基于关联规则挖掘的方法去预测缺陷的关联性和缺陷修正工作量，并将该方法应用在超过15年的200个项目中。他们[191]还提出了一种基于灰色模型GM（1，1）和Verhulst的方法去预测软件物理时间的阶段工作量。在大型的现实世界的软件工程数据集上验证该方法，并且将其与线性回归方法和卡尔曼滤波方法相比，结果显示精确度至少分别提高了28%和50%。

中国科学院软件研究所的李明树等[101]全面回顾、分析了软件成本估算的各种代表性方法，也归纳、讨论了与成本估算强相关的软件规模度量问题，进一步研究了软件成本估算方法的评价标准，并指出软件成本估算方法下一步的主要发展趋势。他们还调研了中国工业界中影响开发工作量的因素，并在140个软件组织的999个项目中验证了6个因素（项目规模、团队规模、工期、开发类型、业务范围、编程语言）对开发工作量的影响。

缺陷预测方面。南京大学的周毓明等[210]利用面向对象设计度量元预测高、低严重程度的缺陷。之后，文献［212］给出了一种新的指标来衡量名称相同的类在两个相邻版本中所发生的变化。他们[109]还采用统计元分析方法在102个Java系统中去探究62个面向对象度量元预测易出缺陷类的能力[109]。他们在研究[186]中还利用方法调用异常来提高基于频谱的缺陷定位技术在面向对象程序中的有效性[186]。南京大学的陈振宇等[123-124]提出了一种基于聚类的策略来确定故障定位的准确性。

中国科学院软件研究所的张文、王青等提出了基于K近邻和社会网络分析的缺陷修复人推荐方法Drex。接着，他们[200]利用LDA主题模型针对不同程度开发人员对于修复缺陷的不同程度兴趣和专长进行建模，并在此基础上提出DRETOM方法以进行缺陷修复人推荐[200]。

西安交通大学的宋擒豹等[214]搜集了类级别上的静态代码度量元和变化数据，并分析发现80%的代码行变化存在于20%的类中，然后他们使用分类技术来预测易改变的类并得到了良好的效果。

南京大学的周志华、清华大学的张洪宇等[102]将主动学习、组合学习和半监督学习

的思想引入软件缺陷预测，针对可利用的软件缺陷历史数据稀少珍贵的问题，提出了基于抽样的主动式半监督缺陷预测方法。

清华大学的张洪宇等[39]提出一种基于调用堆栈重复缺陷报告聚类的方法。他们的研究[213]还参考缺陷报告的文本长度和缺陷报告之间的文本相似性，并提出了一种基于信息检索的缺陷定位方法 BugLocator。文献［58］还提出一种利用简单的用户反馈的交互式缺陷定位方法。进一步，文献［65］分析了一个缺陷报告是真正的缺陷还是一个过时的测试。文献［197］还涉及利用分段和堆栈跟踪分析来提高面向缺陷报告的缺陷定位的准确度。

南京大学的张赫等人[108]采用系统评价的研究方法综合分析了目前 CRC 模型的相关研究[108]。其结果表明，在基于 4 个基本 CRC 模型的 18 个具体的计算算子中，并没有一个相对确定的最好算子，并且，缺乏工业领域的应用也是 CRC 模型相关经验研究当前状态存在的问题之一。

3.2 软件需求、软件结构设计及度量

3.2.1 需求分析

软件需求分析是软件开发活动中的重要阶段，大量研究表明，修复该阶段发生的错误的代价远远高于其他阶段[97]。目前，在需求工程领域的经验研究方面有如下代表性工作。

在需求捕获（Requirements elicitation）方面，Dieste 等学者采用系统评价方法搜索 SCOPUS、IEEE Xplore、ACM Digital Library 以及 Google 数据库，从找到的 564 篇论文中筛选出 26 篇含有需求捕获技术的经验研究。这些研究用来评估和评价 43 种需求捕获技术，并根据获得的结果提出了一整套指导需求捕获实践的原则和建议[42]。有研究者提倡在需求分析阶段应充分发挥创造性以改善需求捕获的效果。Saha 等人搜索 IEEE Xplore、ACM、Compendex、Inspec、Springerlink、Science Direct 等不同的电子数据库，他们根据获得的结果得出创造性在需求工程中所起的作用[152]。

鉴于用户往往是需求分析的主要参与和贡献者，Bano 等人研究用户参与需求分析活动的程度与项目成功之间的关系，他们从找到的 87 个经验研究中（1980 年 2012 年）发现仅有 13 个把重点放在用户参与需求分析活动中[14]，所得结论是，在需求工程阶段有效地让用户参与可以减少其他阶段用户参与的必要性。此外，他们还提出了在需求分析中应该使用的能提高用户参与程度的推荐活动列表。

Svensson 等人发现需求工程研究者在如下 5 个方面进行了经验研究：需求捕获、依赖性、质量需求度量、成本估算及确定需求优先次序。此外，他们还研究了现有的质量需求管理方法的优点及局限性等问题[177]。

Condori-Fernandez 等人采用系统评价方法，收集现阶段软件需求规约技术研究中的哪些方面已经进行了相关的经验研究、研究的背景是什么以及采用了什么研究方法等方

面的资料。他们分析了所找到的46篇相关论文，得出以下结论[38]：大部分经验研究评估的焦点是软件需求规约的可理解性，实验是最常用的研究方法，评估的环境大部分是学术机构。

3.2.2　软件结构设计

软件体系结构是软件系统顶层的设计蓝图，是软件开发从需求到详细设计的中间产品。软件体系结构在软件开发过程的整个生命周期中，对软件质量、开发成本和进度的控制都起到关键作用[15,120]。以软件体系结构为中心的软件开发（Architecture-centric software development）已经成为大型复杂软件系统[15,159]的基石和准则，并得到一系列工业化软件开发过程（CMM、UP）、软件体系结构描述规范（ISO 42010：2011[75]、ADL[119]）、支撑工具（IBM rational software architect）和成熟框架平台（.NET、J2EE）的支持。

软件体系结构是软件开发的中间制品，具有一定程度的抽象性和多样性，一般由架构师来负责，设计过程本身难以完全自动化，因此软件体系结构领域的研究难点在于其方法和效果难以得到有效评价。经验软件工程强调方法在应用中的实际评价，这些了法包括调研、实验和案例研究等，因此它们也适用于软件体系结构领域的研究，并已得到广泛应用[145,48]。

（1）体系结构的系统评价

软件体系结构的概念从1992年被提出以来[141]，经历了20多年的不断发展，已经成为软件工程研究和实践中相对成熟的领域[37]，在软件开发，尤其是大型软件开发中得到了广泛应用。系统评价（Systematic Review）作为经验软件工程中的一种主流文献综述方法，近年来被广泛应用于软件体系结构领域内特定研究主题的系统性研究和实践成果的提炼与呈现上。Williams和Carver对软件系统演化过程中体系结构层面的变化进行了综述研究，将软件体系结构的变化提炼为4种：完善变化、修正变化、适宜性变化和预防性变化，并将它们用于软件体系结构层面的变化分析和处理中[196]。Breivold等则对软件系统演化中采用的软件体系结构方法进行了系统评价，提炼出了主流的基于软件体系结构的系统演化方法，并对所有方法进行了分类和综合分析[26]。Oliveira等对现有面向服务体系结构（SOA）的参考体系结构模型进行了提炼，并提出了一个综合面向服务体系结构的参考模型[136]。Koziolek从软件体系结构对系统的可持续性和生存性影响方面进行综述和分析，提出了一系列保持软件体系结构层面系统可生存的设计准则[94]。Aleti等对软件体系结构的优化方法（尤其是在大型复杂软件系统中，针对复杂的质量需求和目标，如何通过搜索技术得到最佳的软件体系结构设计方案）进行了系统综述，并基于综述结果为软件体系结构的优化指出了若干重要研究方向[5]。Klein和van Vliet对系统的系统（System-of-systems，SOS）领域的软件体系结构研究进行了综述，指出了该领域存在的若干挑战性问题，包括SOS的体系结构可扩展性和稳定性问题[92]。李增扬等对知识方法应用于软件体系结构进行了综述，按照软件体系结构过程，针对知识方法对各个软件体系结构活动的支持进行综述，为知识方法在架构活动中的应用奠定了基础[103]。Shahin等对

可视化技术用于软件体系结构建模和描述进行了综述，提炼出了主流的软件体系结构可视化方法，为软件体系结构可视化实践提供了指南[157]。Tofan 等对软件体系结构中的决策方法和实践进行了系统综述，提炼出各种方法的优势和不足，并为软件体系结构决策的下一步研究提出了可行的研究路线[184]。

（2）体系结构的工业调研

工业调研方法被用来反映当前工业界针对软件体系结构方法、概念的理解和应用状况。如 Malavolta 等对工业界目前使用和需要的软件体系结构描述语言的调研，发现了目前软件体系结构描述语言在工业应用中存在的主要问题，以及可能的应对方法[117]。Tang 等调研了工业界如何理解、使用和归档软件体系结构设计原理，发现了该领域存在的主要挑战，同时提出了未来应解决的问题[181]。Falessi 等比较了现有的用于制定软件体系结构决策的技术，并认为目前没有万能的方案，不同的技术适合解决不同的问题[49]。Lehrig 和 Becker 对软件体系结构的受控实验进行了调研，总结出了相关的经验教训，并提出未来仍然需要实施更多的软件体系结构方面的受控实验[98]。丁炜等调研了 2000 个开源软件项目，并发现仅 108 个项目有软件体系结构的文档，以及在这 108 个项目中模型、系统和任务是最常被记录的内容[43]。Rost 等调研了 147 个工业界从业者在软件体系结构文档方面遇到的问题以及对于未来的期望，指出目前软件体系结构文档更新不及时，以致于和已有数据不一致的问题，同时还发现对目前软件体系结构文档的检索是一个重要的挑战[151]。Tofan 等关注于软件体系结构的决策，通过调研 43 个工业界的架构师，从多个维度比较了这些架构师做出的决策，发现了该领域存在的一些问题[183]。de Silva 和 Balasubramaniam 调研了防止、检测和恢复软件体系结构侵蚀的技术，并对这些技术进行了分类，总结出了各自的优势和劣势[163]。van Heesch 和 Avgeriou 关注于制定软件体系结构决策的推理过程，通过对工业界 53 个架构师的调研，总结出了一些软件体系结构决策推理的最佳实践[68]。

（3）体系结构的受控实验

受控实验方法被用于评价软件体系结构方法在控制条件下对特定变量的应用效果。Javed 和 Zdun 发现了应用软件体系结构的追溯性可以极大地促进用户对软件体系结构层面的理解，同时发现个人经验对于理解软件体系结构并非决定性因素[77]。Lytra 等研究了软件体系结构知识的复用对软件体系结构制定决策过程的影响，并发现复用决策模型可以显著提高新手架构师的工作效率和效果[110]。Shahin 等研究了如何用软件体系结构的设计决策提高用户对软件体系结构的理解，并得到了若干结论，如发现了在软件体系结构文档中使用软件体系结构设计决策并不能降低完成软件体系结构设计任务的时间[158]。De Graaf 等研究了基于本体的归档如何支持用户寻找所需的软件体系结构知识，并发现了对本体的构建如果是以对软件体系结构知识的良好理解为基础，则可以显著提高用户寻找所需软件体系结构知识的效率和效果[60]。Haitzer 和 Zdun 关注于软件体系结构构件的图表对新手架构师理解软件设计的影响，并指出如果构件图表中的元素能直接链接到需要被理解的问题，则用软件体系结构构件的图表能增进新手架构师对软件设计和软件架构的理解[63]。van Heesch 等验证了在恢复软件体系结构决策时应用软件体系结构模式

可以使软件体系结构决策恢复得更加有效这一假设，并得出通过应用软件体系结构模式可以显著提高决策质量这一结论，但没有证据支持用这一模式可以影响决策的数量[69]。Ferrari 等识别了一些会被已有软件体系结构影响的需求特征，以及影响程度，同时还识别出了影响这些需求特征的软件体系结构的元素[51]。Babar 等关注于在评价软件体系结构时对场景简介的开发，指出分布式的会议更加有利于场景简介的开发，但是这在很大程度上依赖于工具的支持[12]。

（4）体系结构的案例研究

案例研究方法被广泛应用于实际应用环境中对软件体系结构方法的评价。Moreno-Rivera 和 Navarro 对软件产品线体系结构应用于 Web 地理信息系统开发的方法进行了研究，发现该方法能够满足软件产品线体系结构设计中的关键性系统质量需求[128]。López 等提出了一种基于网页的语义方法和工具（Toeska/Review），以比较和显示软件体系结构的原理，并在博物馆集成项目中（Contexta）使用[111]。Buckley 等评估了专家架构师使用 Reflexion Modelling 方法时软件体系结构的一致性，并有针对性地对该方法进行了扩展[28]。Schultis 等识别了 3 个协作模型以及一系列软件体系结构的挑战和问题，并对这些挑战进行了分类[154]。Brunet 等分析了为期 5 年的 Eclipse 体系结构检查报告，识别出一些对开发者有用的规则，发现了代码是如何偏离期望的软件体系结构以及开发者是如何处理这些问题的[27]。De Graaf 等提出了基于本体的软件体系结构归档方法，总结出 8 个可以影响本体成功构建的因素，并通过案例描述该方法是如何获得的，并对软件体系结构的知识需求建模[59]。李增扬等提出了基于软件体系结构决策和变化场景的识别软件体系结构技术债（Architectural Technical Debt）的方法，并评估了该方法的有效性和可用性，发现该方法非常适合在工业案例中使用且对软件体系结构技术债的识别有效[105]。李增扬等建议使用软件模块化度量标准来表示软件体系结构技术债，通过验证 ANMCC（Average Number of Modified Components per Commit）与模块化度量标准的联系，发现了两个模块化度量标准与 ANMCC 紧密关联，并可使用于软件体系结构技术债中[104]。Díaz 等验证了一种用于敏捷构建和演化产品线体系结构的方法，并指出该方法能有效地促进从业者构建和演化产品线体系结构[46]。Etemaadi 等研究了元启发式优化方法是如何优化软件体系结构设计过程的，并通过案例发现用该方法可以找到满足所有质量属性和约束的有效解决方案[47]。Mattsson 等报告了模型驱动开发在真实项目中的使用情况，发现了模型驱动开发在建模软件体系结构的设计规则方面的不足，并提出了相关建议[118]。丁炜等通过分析开源项目中的开发邮件列表，发现邮件列表中的信息能将软件体系结构的变化反映到代码层面（但是大部分软件体系结构的变化都是预防性的），同时指出开源项目的软件体系结构在第一个稳定版本后趋于稳定[44]。Nakagawa 等研究了软件体系结构如何直接与开源软件质量联系起来，并提出了使用软件体系结构重构来修复软件体系结构的方式[131]。Feilkas 等通过研究软件体系结构知识的缺失问题来回答软件体系结构在多大程度上与代码一致；文档是否能够反应预想的软件体系结构等问题[50]。

3.2.3　国内研究现状

武汉大学的梁鹏、李增扬等对知识方法应用于软件体系结构进行了综述，按照软件

体系结构过程，对知识方法针对各个软件体系结构活动的支持进行综述，为知识方法在架构活动中的应用奠定了基础[103]。他们还提出了基于软件体系结构决策和变化场景的识别软件体系结构技术债的方法，并评估了该方法的有效性和可用性，发现该方法非常适合在工业案例中使用且对软件体系结构技术债的识别有效[105]。接着，他们还建议使用软件模块化度量标准来表示软件体系结构技术债，通过验证 ANMCC 与模块化度量标准的联系，发现了两个模块化度量标准与 ANMCC 紧密关联，并可使用于软件体系结构技术债中[104]。

武汉大学的丁炜、梁鹏等调研了 2000 个开源软件项目，发现仅 108 个项目有软件体系结构的文档，以及在这 108 个项目中模型、系统和任务是最常被记录的内容[43]。他们还通过分析开源项目中的开发邮件列表，发现邮件列表中的信息能将软件体系结构的变化反映到代码层面（但是大部分软件体系结构的变化都是预防性的），同时指出开源项目的软件体系结构在第一个稳定版本后趋于稳定[44]。

3.3 软件测试与质量保障

3.3.1 软件测试

近年来，经验方法在软件测试领域的研究和应用方面呈现越来越多的趋势，以 ICSE2015 大会为例，本次大会中在测试方面共有 16 篇论文，其中 12 篇均为采用实验分析的研究论文。软件测试的实验过程相对统一：选择实验程序、运行测试用例、收集测试结果、进行实验统计分析。

在软件测试实证分析中，影响结果的因素主要包括：实验程序（含测试用例）、评价标准（覆盖、变异、故障等）和统计方法（描述统计、推断统计）。我们以 ICSE2015 这 12 篇论文为例，按这 3 个因素分别讨论软件测试实证分析的现状。

（1）实验程序

在 12 个研究中，所有实验均采用开源实验程序。其中，6 篇论文采用 C 语言程序，5 篇论文采用 Java 语言程序。实验程序的来源主要是 SIR、GNU 及其他。50% 的实验都采用了 SIR[164] 上的程序作为实验对象。实验对象的规模则相差较大，从几十行到十万行代码不等。值得提醒的是，文献虽采用了较大规模的实验程序，但其测试主要基中于类这一级别，因而实际程序规模大幅减小。

实验程序的数目差异也较大。只有两篇论文[40,198]选用了超过 30 个实验程序，满足基本统计数量要求。7 篇论文只采用了数量极少（10 个以内）的实验程序。现有论文在实验程序的选择上没有严格策略，基本采用前人常用的说辞。过少的实验程序数量和随意的选择策略，使得实验结果的外延风险增加，从而限制了实验结论的适用范围。测试用例是软件测试实验的必要组成部分。SIR 自带测试用例集，实验重现方便，因而广受研究人员欢迎。除此之外，采用工具产生测试数据也是一种常用手段，但这一手段通常局限在规模较小的白盒测试数据生成工具中，如 KLEE、EvoSuite、Randoop 等。人工生

成测试数据实验成本高，因而难以被广泛采用。综上所述，实验程序开放，但相应测试数据并未开放，这对研究论文的实验重现带来较大的困难。

（2）评价标准

软件测试的最主要目标是发现失效行为，即俗称的 Bug。因而 Bug 检测能力成为软件测试方法的常用评价标准之一。12 篇论文中，有 3 篇采用真实 Bug 进行软件测试方法的评价[140,182,198]。然而，我们不难发现，真实的 Bug 数量较少，因此难以提供有效的统计支撑。

在 SIR 的程序库中，大部分采用人工注入 Bug 作为实验程序。人工注入 Bug 的一种常用手法是变异，即通过机械式的语法改变进行 Bug 注入。变异分析存在成本昂贵和等价变异等困扰，使其难以在工业中应用，但在软件测试实证分析研究中，仍然不失为一种好的评价手段。其中文献［40］采用了变异分析方法。

在难以获取 Bug 信息的情况下，控制流覆盖和数据流覆盖是一种替代的软件测试评价手段。12 篇论文中，有 5 篇采用了覆盖信息作为评价标准[53,140,194,198,178]。尽管基于覆盖的评价标准在学术界和工业界都得到了广泛采用。然而，近几年开始出现了批评的声音：为了覆盖而覆盖没有实际的 Bug 检测能力。

时清凯等提出一种距离熵的测试用例充分性准则，以期通过度量测试用例集的多样性来刻画其 Bug 检测能力[161]。这一基于软件行为的度量方式为软件测试能力评价标准提供了新的思路。

（3）统计方法

统计方法是实证分析的重要组成部分，通常分为描述统计与推断统计。描述统计通常采用统计图表、汇总数据等方式，用于直观展现实验结果的基本情况。均值和百分比是最常见的数据汇总方式，几乎出现在所有论文中。箱形图易于呈现实验结果的平均值和波动情况，散点图和折线图易于呈现实验结果的趋势走向和相关性，被广泛采用。

推断统计是利用样本数据来推断总体特征的统计方法。显著性检验是软件工程中常用的推断统计方法，包括 t－检验和 U－检验等。

3.3.2　质量保障

软件质量保障技术除了上述的软件测试之外，还包括静态程序分析、形式化规约、统计过程控制、软件评审等多种技术手段。在这些技术手段中，软件评审一直是经验软件工程研究的热点。软件评审一般是指应用人工阅读的方式静态地评价软件开发产物，以期尽量在项目早期发现并且消除缺陷，从而提升最终软件产品的质量。典型的评审对象包括各类开发文档和代码。评审的方式也有多样，例如，个人阅读、走查（Walk Through）、正式评审（Formal Inspection）等。其中，正式评审自从 1976 年由 M. Fagan 引入软件开发以来，在工业界和学术界都引起了很多关注。Zhang 等人的调查表明，正式评审是软件工程实践中被研究和报告最多的软件工程实践之一[207]。应用经验软件工程方法研究软件评审的实践主要包括如下一些具体的研究方向。

（1）阅读技术

阅读是软件评审最基本的实践之一，相应地，各种阅读技术（Reading Techniques）也是软件评审相关研究中最活跃的研究课题之一[153]。目前被识别的主要阅读技术包括：基于检查表的阅读（Checklist-based Reading，CBR）、基于场景的阅读（Scenario-Based Reading，SBR）、基于缺陷的阅读（Defect-based Reading，DBR）、基于视角的阅读（Perspective-based Reading，PBR）、基于用例的阅读（Usage-based Reading，UBR）、抽象驱动阅读 Abstraction-driven Technique（ADT）、任务驱动阅读（Task-driven Inspection，TDI）等。当然，还有最基本的随机阅读（Ad Hoc Reading，AHR）。

经验软件工程在这个方面的主要研究集中在针对不同的评审对象，比较不同阅读技术在缺陷发现率、缺陷发现成本等问题上的差异，从而试图找到一个具体场景下的较为合适的阅读技术。

例如，Caroline D. Rombach 等人通过软件工程实验对比了在 ER（实体关系）模型的设计文档评审中，用什么样的阅读技术可以更高效地发现 ER 模型中的缺陷。Thomas Thelin 等人则通过一个实验，比较了 UBR 和 CBR 阅读技术在需求评审中发现缺陷的时间效率和缺陷发现率上的差异。Maldonado 等人通过实验比较了 PBR 和 CBR 在发现缺陷上的差异[81]。Oliver Laitenberger 等人通过准受控实验，比较了 PBR 和 CBR 对于代码评审在缺陷发现率上的差异[135]。Alastair Dunsmore 等人通过实验，研究了针对面向对象代码的评审的 3 种阅读技术，即 CBR、UBR 和 ADT[2]。

值得注意的是，Ciolkowski 等人的调查研究表明，在工业界，大部分评审者（60%）都缺乏必要的评审经验。在剩下 40% 的评审者当中，仅仅只有 12% 的人接受过正式的评审训练[36]。因而，评审新手是一个普遍存在的现象。Rong 等人以软件企业中的评审新手为研究对象，对比 CBR 和 AHR 这两种不同的阅读技术，结论表明，就评审效果而言，两者没有显著差异，但是 CBR 可以提供更好的基础，有助于未来改进[150]。

（2）评审效果的影响因素

评审效果的影响因素相关的经验研究也非常丰富。一般而言，对于评审效果主要通过两个指标来衡量：即功效（一次评审发现的缺陷总数）和效率（单位时间内发现的缺陷数量）[189]。目前经验方法研究所识别的主要因素包括：个人能力、评审速度、评审过程、准备阶段、材料总量、团队规模、人员培训、知识背景和经验、评审轮数以及材料类型等。通过这些研究，建立起对于影响评审效果的因素的相对丰富的理解。

例如，Biffl 等人通过实验并引入成本效益（Cost-Benefit）模型分析了阅读技术、评审团队规模，评审时间（速度）等因素对缺陷发现率和成本效益比的影响。研究表明，结合不同的阅读技术比单独采用最好的阅读技术拥有更高的效率；更好的评审过程比额外添加评审人员更有效；额外地增加评审人员，并不能线性地提高评审效率。因此，为了实现较高的缺陷发现率所要付出的开销远大于平均水准的缺陷发现率对应的开销[171]。

Albayrak 等人通过实验研究了个人的教育背景、经验、英语水平、公司等因素对于需求评审的影响。实验结果表明，在进行需求评审时，没有软件工程背景的评审者效率甚至更高；拥有更多的需求经验的评审者效率更高；评审者的英语水平和评审效率正相

关；评审者的公司背景对评审者评审效率有一定的影响[3]。

Sauer 等人基于行为学的理论进行了经验研究，研究指出，评审者个人的专业水平是影响评审效率的最重要的因素；提高评审效率有 3 种有效的方式，即，选择擅长发现缺陷的评审者、通过训练提高个人专业水平、选取适当的评审人数[33]。

Kemerer 等人通过对个人软件过程（Personal Software Process）的数据研究，发现对于个人评审来说，在 200 LOC/hour 或稍低的评审速度下，评审效率最佳[34]。

（3）支持工具的效果

工具支持主要是研究应用于软件评审过程中辅助评审者完成评审的软件系统和工具。该领域的研究起于 20 世纪 90 年代，也正是互联网应用迅速发展的时期。该领域的研究主要集中在两个方面：①研究针对不同的评审文档，工具支持的评审（Tool-based）和纸质评审（Paper-based）的差异；②为特定目的，提出新的工具，并进行验证。

Halling 等人通过 3 个独立实验（参与者、实验设计和执行均类似）比较了针对需求文档的工具支持评审和纸质评审这两种评审方式，得出以下结论：工具支持评审和纸质评审的缺陷发现率相近；工具支持评审可以降低所发现缺陷的重复率；此外，工具支持评审可以提高单位时间内发现的缺陷数[61]。

Biffl 等人提出了 GRIP（Groupware-supported Inspection Process）框架并通过实验比较了在 GRIP 上进行评审和纸质评审的差异，证实了相较于纸质评审，GRIP 能够提高评审效率。GRIP 可以为评审团队提供一个框架并整合多种工具，它支持独立发现缺陷、评审会议以及评审管理等功能[172]。

Perry 等人提出了 HyperCode（通过浏览器使用的评审工具），该工具可以帮助分散在各地的评审者进行非会议形式的评审。并通过实验证实了，相较于现存的纸质评审，HyperCode 的经济性明显更好；采用 HyperCode 可以缩短评审间的间隔；相较于现存的纸质评审，采用 HyperCode 可以发现更多的缺陷；此外，采用 HyperCode 之后，单个缺陷的描述更加清晰[41]。

（4）其他

在评审子课题下，还有一些其他相关的经验软件工程研究课题，例如：Halling 等人通过一个经济模型研究了评审为软件开发带来的收益，其主要发现是，对于需求协商模型，评审是一种经济的验证技术[62]。Ciolkowski 等人通过两个调查研究了软件评审实践在工业界中的应用状态，研究表明，在软件开发项目中，评审的目的从早期发现缺陷到使团队更好交流，各不相同。此外，目前评审中的阅读方法很多，但是，非系统化的阅读技术应用得最为广泛[36]。

Denger 等人通过实验比较代码评审和功能测试这两种缺陷发现技术对软件开发中的重用组件的不同类型缺陷的发现率，根据实验结果发现，这两种技术对于变异缺陷（Variant-specific Defects）的识别率都很低[35]。

3.3.3　国内研究现状

正式评审自从 1976 年由 M. Fagan 引入软件开发以来，在工业界和学术界都引起了很

多关注。南京大学的张赫等人的调查表明，正式评审是软件工程实践中被研究和报告最多的软件工程实践之一[208]。南京大学的荣国平等人以软件企业中的评审新手为研究对象，对比 CBR（基于检查表的阅读）和 AHR（随机阅读）两种不同阅读技术，相关结论表明，就评审效果而言，两者没有显著差异，但是 CBR 可以提供更好的基础，有助于未来改进[150]。南京大学的陈振宇等提出一种距离熵的测试用例充分性准则，以期通过度量测试用例集的多样性来刻画其 Bug 检测能力[161]。这一基于软件行为的度量方式为软件测试能力评价标准提供了新的思路。

3.4 软件开发与过程改进

软件过程是软件工程领域的一个重要组成成分，对于软件开发的进度和质量都有着重要的影响。在过去的几十年里，软件过程中产生了大量的方法和技术。具有代表性的是基于计划的软件过程方法，与之对应的是敏捷开发方法。敏捷开发由于其不确定性，很多实践者往往从个人的经验和观点出发进行争论，缺乏严谨的科学基础。也有很多学者和实践者并不认同敏捷软件开发，他们认为没有足够的科学证据支持敏捷开发所宣称的益处。经验软件工程方法通过经验研究来确定敏捷软件开发的有效性和局限性，同时也展示当前敏捷软件开发在工业界的使用情况，以供实践者参考，为其科学的决策和改进提供了有效的技术与方法。

3.4.1 敏捷方法的经验研究

伴随着敏捷软件开发本身的发展，对于敏捷方法的研究，早期集中在 XP 方法，后来扩展到 Scrum 和 Kanban 方法。

Dyba 和 Dingsoyr 在 2008 年总结了当时在敏捷软件开发中进行的经验研究情况。在 2005 年以前，共有 36 篇关于敏捷软件开发的经验研究论文，其中有 3 篇是二手研究，33 篇研究中有 25 篇是关于 XP 方法的。Dyba 认为敏捷软件开发领域的经验研究可以分为①敏捷方法的引入和采纳；②人和其社会性因素；③客户和开发者感受；④比较研究。作者认为这些研究确定了一些敏捷软件开发的益处和局限性，但是这些经验研究提供的证据强度都比较弱，不足以提供非常有效的建议供工业界使用。

Scrum 方法是当前应用最为广泛的敏捷方法。文献［222］讨论了 Scrum 在全球软件开发（GSD）中的情况，认为①对于在 GSD 的软件项目中使用 Scrum 的原因和动机缺少研究；②当前 GSD 中项目团队使用 Scrum 实践的情况和本地使用有所不同；③GSD 团队使用 Scrum 时通常会面对交流、合作等问题；④Scrum 实践需要扩展和修改来支持 GSD。

对于敏捷软件开发，很多人都认为其不适合于①大型项目和组织；②地理位置分散的团队；③对于软件质量和安全要求很高的项目。一些研究通过经验方法研究并介绍了在大型组织中进行敏捷软件开发的有效途径。

全球软件开发对于当前世界的全球化进程和软件外包产业非常重要，但是进行全球化开发会面临语言文化差异、信任问题、跨时空交流沟通等问题。一些研究通过经验研

究的方法讨论了如何在分布式团队环境下进行敏捷软件开发。

敏捷软件开发提倡更少的度量，研究发现使用度量的行为主要集中在以下领域：sprint 计划、进度跟踪、软件质量度量、修复软件过程问题、激励开发者。此外发现，尽管敏捷团队使用很多敏捷文献中建议的度量，但是他们同样也采用很多团队定制的度量。另外，最有影响的度量是开发速度和工作量估算。文章发现，度量在敏捷开发中的使用和传统软件开发相似：项目和 sprint 需要被计划和追踪；质量需要被度量；过程中的问题需要被识别和解决。

3.4.2　敏捷实践的经验研究

经验研究方法已被研究界广泛应用于研究敏捷实践。经验研究对于敏捷实践的研究主要集中于质量检查和结对编程（Pair Programming，PP）中。经验研究分析了大量的工业界项目，结果表明，在敏捷实践中：时间盒、计划会议、学习循环、演进和分级需求、每日讨论、产品愿景等实践被广泛采用，其他采用较多的有客户参与、质量检查。同时，结果还表明，敏捷实践已经在自动化、公司管理、咨询、金融、政府、互联网、医疗、媒体、通信等多行业的软件开发中被广泛采用。

软件质量是软件产品中非常关键的问题之一。测试驱动开发（Test- Driven Development，TDD）由 Kent Beck 在 2000 年提出，它是敏捷软件开发的一项基本实践，同时也是极限编程中非常重要的一部分。测试驱动开发方法要求开发者在编写具体功能前先编写测试代码，再编写功能代码。这种方法驱动着代码的设计与功能的实现、测试以及驱动代码的重构。虽然 TDD 的问世时间并不长，却在学术界引起了广泛研究。长期以来，在 TDD 经验研究中，实验和案例分析这两种经验研究方法被广泛采用。

经验研究对 TDD 的研究目标主要集中在 TDD 的有效性和适用性方面。有效性研究的主要目标为：对软件质量的影响、对生产率的影响、对软件设计的影响、对软件维护的影响。

敏捷需求工程被用来定义敏捷的计划、执行和解释需求工程活动，此外，以协作为中心的敏捷方法中的需求工程活动所引起的问题仍然没有被广泛认知。

3.4.3　软件过程改进

软件过程改进是一个实践和研究并重的课题。各种改进模型的提出、裁剪和应用形成了这个领域大部分的研究工作，具体包括以下几个典型的方向。

软件过程理论兴起于 20 世纪 80 年代，经过不断的发展和完善，已经形成一套比较成熟的理论。一些典型的模型也得到了广泛的应用。经验软件工程在过程模型方面的研究主要是各类软件过程改进（Software Process Improvement，SPI）模型在不同规模的公司及组织中的适用性，提高软件产品的质量或者加速软件开发过程。

Staples 等人的案例研究表明，CMMI 模型作为一个软件过程改进框架，不仅可以让一些大型组织通过达到高级别的成熟度获得高生产率和产品质量，同时 CMMI 也可以很好地适用于中小型软件企业[170]。Niazi 等人在 11 个企业访谈结果的基础上，设计并且验证了一个软件过程改进的模型[134]。将 CMM/CMMI 和其他技术融合也是此类研究的关注

重点，例如，Park 等人研究了 Six Sigma 在 PSP/TSP 过程中的应用，结合实例证实，Six Sigma 可以使软件工程师更好地分析 PSP 数据，并系统地改进过程性能[139]。

在软件过程管理中要考虑很多关键问题，以对过程改进展开有效的建模和评估。然而，庞大的元素数量和多样性使得这种考虑面临很多挑战。García 等人提出并且验证了一种 FMESP 框架，即过程和测量软件建模的集成管理框架[54]。每一个过程改进项目都必须优先选择并集中于过程的特定方面。此外，进行每一个此类选择都需要沟通、激励所有的利益相关者（管理以及参与或受影响的各方）。Birkholzer 等人提出了一个概念框架和工具集，结合实例分析，证实该框架可以用于目标驱动的搜索策略，以及特定场景的仿真分析[20]。Jedlitschka 提出了一个基于经验改进的集成软件过程改进框架，并在工业界和学术界分别开展了案例研究和受控实验研究[78]。

某些时候，全面实施 CMMI 框架看起来负载过重，对此，Sivashankar 等人提出并且验证了一个框架来更好地应用 CMMI 模型，这有助于实现 SPI 与一些标准的实施[162]。Harjumaa 等人介绍了一组模式，以指导小公司软件检验过程的改进，初步的实验结果表明该模式提供了一种可行的快速启动 SPI 活动的方法[66]。

Bayona 等人提出涉及流程改进的关键成功因素包括：承诺、精准的业务战略和目标、培训、交流、资源、技能、员工参与、改进管理、定义过程、软件过程改进流程的监测、变化管理、文化、政策、角色和责任、工具和指导的规定，以在技术上指导软件过程改进的方法。此外，在此模式下，由于员工参与过程改进活动，所以可以最大限度地减少改进阻力，保持员工的积极性。当然，过程改进计划还需要资源、人的技能、能力和知识来执行活动[17]。Georg Kalus 等人以过程裁剪的参考标准为依据，应用系统评价方法研究，得出了类似的结论[56]。Rong 等人进一步提出这些裁剪的标准必须是在一定上下文之下才有意义，因而，在应用经验软件工程方法进行相关研究的时候，必须定义具体的上下文。Michael 等人则提出定义、选择和验证措施可能比 SPI 评估更有意义等[125]。

Beijun Shen 等人以一个小软件公司为背景，提出了一种软件过程改进的实施方式，实施措施主要包括过程建模、过程自动化，以及过程测量。Serrano 等人同样描述了在一个小的软件开发公司的过程改进经验中，使用团队软件过程（TSP）自主地实施软件过程改进，而在组织层，应用软件能力成熟度模型（CMM）以及 IDEAL 模型作为组织改进模型[156]。Sulayman 等人应用系统评价方法研究软件过程改进在中小型网络公司中的应用。这些中小型公司往往具备类似特点，例如，严格的预算约束、严格的期限和短期战略要求等[176]。

应用支持工具能够很好地支持、管理并规范软件过程，因此应用支持工具研究在软件工程改进相关的研究中也是数量较多的一类。

García、Du 等人报告了一些案例研究，目的是探讨和评价支持工具对过程改进的作用[55,45]。

3.4.4 国内研究现状

上海交通大学的沈备军等以一个小软件公司为背景，提出了一种软件过程改进的实

施方式，这类方式结合了灵活和控制的考虑，并且不妨碍一个小公司的创新性质，实施措施主要包括过程建模、过程自动化，以及过程测量[223]。

中国科学院软件研究所的王青等基于软件项目管理和过程改进工具 Qone 在使用中产生的数据，通过进行后续的文件调研和访谈，详细分析了软件过程管理工具的使用情况以及带来的启示，例如该工具的作用随着任务的类型变化而变化；对细粒度的任务更容易进行预测和控制[45]。

3.5　开源与分布开发

在 20 世纪末，开源软件系统取得了巨大成功，已经成为软件技术创新和产业发展的主要模式之一，展现出社会、经济、组织与管理、技术、实践等方面的重要属性。这为寻求高效的软件开发方法提供了一种用户创新驱动、成本低、质量高的新思路。为探索高效的软件生产方法，有很多的报告致力于研究开源实践。同时，开源软件的数据开放性也为经验研究在此领域的发展提供了条件。目前，针对开源软件开发的经验研究主要存在两种类型：一类是基于对各类开发数据的分析与挖掘，发现模式；另一类是提出新的理论或模型，并使用开源经验数据进行验证。

开源已经成为软件技术创新和产业发展的主要模式，展现出社会、经济、组织与管理、技术、实践等方面的重要属性。从经济学角度，研究者关注的核心问题之一是：为什么会有大量的个体自愿且无偿地参与到开源软件开发中。从组织学的角度，研究者则关注：为什么一个看似松散无序的群体能够生产出高质量的软件制品。从技术的角度，研究者则关注：开源软件开发背后蕴含/基于的方法和技术是什么，使用了哪些支持工具。从实践角度，研究者则关注：商业型组织如何有效参与到开源软件开发中。

开源实践的一个重要特点是全球分布式开发和大众化协同。分布在世界各地的开发者协同发布一个可用的高质量软件时，会面临大规模的通信、协调和合作的挑战，这体现了社会化开发的复杂性。如何利用软件社区中的海量软件数据和丰富的软件知识，如何度量对程序员的技能、成长途径、他们跟环境的交互、环境对他们的影响，如何进一步理解群体构造的机理和演化规则，如何解决社会化开发的可知与可控难题，这些在目前都受到了广泛关注。

3.5.1　开源软件的开发和演化

Scacchi[154]基于对已有经验研究的回顾，研究开源软件是如何开发和演化的，主要包括：为什么人们会参与开源软件开发，资源和能力支持的开发过程，在项目中如何进行协同和控制，项目内部的信息沟通和社会网络，开源软件作为多项目的软件生态系统，开源软件作为社会运动，并且发现开源软件研究中未来的机会。

软件演化在开放的软件开发中面临着越来越多的挑战，涉众的多元化、开发的社区化、环境的开放性都给软件的演化带来了新的问题，Scacchi[154]发现开源软件项目在进行演化时，不是依赖于用户提出的形式化的功能性需求文档，而是依赖于在线制品数据，

例如需求请求、论坛中的信息、聊天脚本等。Pagano 等研究 AppStore 中的用户评论，发现这些评论内容包括软件需求和用户体验，例如已有功能的不足、需求请求、功能的使用场景等，但是这些评论的质量却差别很大。他们指出如果有工具或方法来系统地过滤和聚类这些用户评论，可以节省项目开发人员的工作量，从而有效地指导软件演化。为了节省花费在收集和理解用户评论上的时间，Guzman 等[224]提出了一种自动化方法，以帮助项目开发人员过滤、组织和分析用户评论。该方法首先用自然语言理解的方法识别评论中的细粒度的需求信息，然后抽取出有这些需求的用户的偏好并且进行打分，最后用主题模型技术将细粒度的需求组织为有意义的高层需求集合。类似地，Chen 等[225]开发了 AR-Miner，以过滤掉噪声评论、抽取信息量大的评论、对评论进行聚类以及优先级排序，从而指导软件演化。

Zimmermann 等[215]利用版本控制系统数据来寻找软件代码中的变化模式（Change Pattern），进而对软件维护中代码的修改提供自动的启发式提示。依靠版本变动历史信息构建事务，并在事务基础上进行挖掘以构造代码维护的规则集合，进而利用规则集合来对代码维护进行指导。Howison 等[71]研究了使用 SourceForge 数据挖掘可能遇到的问题和易犯错误的项目，作者指出使用 SourceForge 中开源项目数据时所需要注意的一些事项，包括在获取数据、分析数据时需要考虑哪些问题等，为利用开源项目数据提供了有益的指导。Fischer 等[52]以 BSD 产品族（FreeBSD、Mac OS 等）为实验对象，突破了以单个项目历史信息为研究对象的局限，研究对一个产品族历史信息的挖掘和利用，结果表明，用他们的方法可以发现不同项目之间的代码依赖关系。Huang 等[72]研究了如何对版本控制系统信息进行分析整理，为新进入开源项目开发的项目外围人员提供最初的学习指导。

重构是开源软件演化的重要方式，Ratzinger 等[149]通过对开源软件的分析发现开发的生命周期属性与重构有关，因此提出了一种通过挖掘软件的演化信息来预测重构的方法。Taneja 等[180]提出了用 refacLib 工具来自动地探测软件库中的重构，通过对重构前后两个版本的代码的语义分析和启发式的方法探测重构。最后通过对 5 个开源软件库的重构探测验证了方法的有效性。Liu 等[107]提出了一种重构的调度方法，并利用开源项目的数据来验证方法的有效性。Palomba 等[137]提出了一种基于代码变更历史信息来探测 code smell 的方法，并通过 8 个开源项目进行验证。Bavota 等[16]提出了基于关联的主题模型的重构推荐方法。

为了研究重构是否能提高软件的质量，Alshayeb 等[6]通过经验研究得到重构并不一定能够提高质量属性的结论。Murphy-Hill 等[130]分析了重构工具的使用和代码中的重构类型，及重构的分布情况。结果表明重构在项目中发生得非常频繁，大部分重构都是零散地分布在其他开发活动中，但是重构工具很少被使用。Rachatasumrit 等[146]分析了重构对回归测试的影响，结果显示 22% 的重构在回归测试中被测试了，并且有一半失败的回归测试涉及了重构。Murgia 等[129]分析得到了代码之间的 fan-in、fan-out 与重构行为的相关关系。

3.5.2 开放社区的任务分配和调度

在开源软件开发过程中，群体智慧的来源是开源社区的开发者，开发者在开源项目

中的角色、能力和开发者之间的合作网络影响着软件开发的效率和质量。为了分析网络中人员的重要性，Lim 等[99]通过建立涉众之间的网络，并分析网络度量指标来为涉众进行优先级排序。Yu 等[204]提出了一种模型来描述开源软件中开发人员的交流，并利用数据挖掘技术来分析开发人员在组织中的角色。该方法通过对两个开源项目的分析验证了方法的有效性。Soh 等[166]通过对 Mylyn 中记录的操作交互的历史信息，来分析开发人员在进行任务维护时的工作量。Blincoe 等[21]分析任务间的依赖关系与合作关系，并且发现 17% 的任务需要合作。

为了探索开源软件社区中是否存在子社团，以及这种结构是否有利于用户更好地理解自组织的软件社团的工作，Bird 等[23]通过对 Apache 的开发邮件和软件资产库中的活动分析，重新组织志愿者的网络。然后利用复杂网络理论评价该网络并观察其随时间的变化。研究结果显示，在 Apache 中确实存在一些子社团，并且项目的年龄与这些子社团的精密程度有关联。Xu 等[201]通过对 SourceForge 里的开发人员网络进行拓扑分析和演化统计分析，来了解开源软件的现象。研究结果显示，开源软件中的开发人员网络是一个无标度网络。

在开源社区任务分配和推荐方面，Kagdi 等[84]提出了一种为软件变更推荐开发人员的方法。该方法根据历史变更代码提交记录，分析变更经验、经历，以及开发人员的贡献，并以此推荐相关的开发人员。在并行的软件开发中，尤其是开源社区中分布式异地的群体开发中，尽管有事先的沟通和协商，但冲突的变更依然会存在。Kasi 等[86]分析了开源项目中同时变更代码的冲突，并提出了一种最小化冲突的调度方法，最后用开源项目进行验证。

软件缺陷的修复是一项困难、成本高昂且漫长的过程，快速准确地将缺陷报告分派给合适的开发人员可以大大降低缺陷的修复时间。

Xie[200]等提出了一种基于主题模型的软件缺陷修复人员推荐方法（DRETOM），该方法利用开发人员参与缺陷修复的历史活动数据，对缺陷报告进行主题建模，并进一步计算开发人员在这些主题上的兴趣度和能力经验值及其参与修复新缺陷报告的概率，最终推荐排名较高的开发人员作为潜在的修复人员。

Xuan 等[202]提出基于开发者排序的缺陷报告分派方法，该方法首先根据开发者在缺陷跟踪系统中共同评论缺陷报告的信息构建开发者社会网络，然后根据开发者的贡献对开发者进行排序。首先使用传统的机器学习缺陷报告分派方法给缺陷报告推荐 N 个修复人，然后根据开发者在开发者社会网络中的顺序重新对这 N 个修复人进行排序。Naguid 等提出了一种基于开发人员历史活动集合的缺陷报告分派方法，他们将开发人员在社区中不同模块里的活动分为“review”、“assign”和“resolve”3 种类型。首先计算出某个开发者在某个模块上这 3 种活动的不同概率。当某个模块上出现新的缺陷报告时，用该方法结合开发者历史活动，推荐一组“resolve”概率最大的开发人员来修复缺陷报告。Wang 等[192]提出一种基于异构网络模型的缺陷报告分派方法（DevNet），用以表征缺陷数据库中开发者之间的多种不同类型的协作关系，并在此基础上结合历史缺陷报告修复信息，将新缺陷报告准确地分派给合适的缺陷修复人员。

3.5.3 开源社区的人员成长

开发者加入开源社区一般都会遵循社会化过程，也称为“洋葱模型”。新加入的开发者首先通过邮件列表的讨论和缺陷跟踪等活动，锻炼自己的能力，在社区中获得声誉，然后逐渐成为核心成员，从而可以直接修改代码和进行设计决策。

项目参与者成为核心代码贡献者是他的技术贡献和社会交互的双重作用结果。Gharehyazie 等[57]认为，对于技术贡献的判断，需要提取不同来源的补丁提交数据，这个较难获取，而社会交互数据较易得到，因此，他们意在研究只基于社会交互数据，能够多大程度地对开发者状态进行建模。通过基于 6 个 Apache 开源项目的研究，发现用社会交互数据，特别是双向的交流数目，可以显著地预测开发者是否能够成为核心代码贡献者。该研究还描述了在成为核心代码贡献者的临近时间点上，开发者的社会交互模式。

当新手加入开源项目时，他们进行适当的培训，从而使其了解该项目的技术问题和组织结构。Canfora 等[29]提出 Yoda 模型，基于邮件列表和版本控制系统，为开源软件项目的新手识别和推荐导师。Yoda 模型在 5 个开源软件项目上被验证，该研究还通过调研这些项目的开发者来了解指导活动是否真的在这些项目中存在。Steinmacher 等[173]研究阻碍新手进行第一次贡献的社会障碍，具体通过系统文献调研，开源项目开发者的问卷调研，以及半结构化的访谈，提出了一个包含 58 个障碍（其中有 13 个社会障碍）的概念模型。新手面临的社会障碍包括：接待方面（例如没有收到回复）、交流方面（进行无用的评论、羞怯等）、方向指导方面（例如如何找到导师）、文化差异方面（例如发送的消息看起来显得粗鲁）等。

3.5.4 开源社区的协作模式及影响

在分布式开发环境中，协作和交互对于完成复杂交错的任务是至关重要的。协作和交互的模式及效率会对开发者的生产率以及软件产品的质量产生影响。

Pinzger[144]等基于开源软件的版本库，用社会网络分析方法，研究开源软件的沟通和协作，并开发了 STNA-Cockpit 工具，该工具不仅可以提供元模型来表示沟通和协作，并且能够用图形可视化技术来发现元模型的实体，由此得到在某段时间内项目的动态变化。

Bettenburg 和 Hassan[18]研究软件开发中的社会交互活动（Social Interactions）对软件质量的影响。该研究基于问题跟踪系统的讨论信息，分为三个部分，一是基于讨论信息得到的软件开发的社会结构（例如开发者在社会网络中的中心度）对软件质量的影响；二是讨论的内容和属性（例如所提交的补丁包含的文件数目）对软件质量的影响；三是讨论活动反映出的团体的工作流（例如缺陷报告中的工作流活动）对软件质量的影响。

项目团队的结构对于项目的表现有何影响？项目团队内部和外部开发者之间的沟通是如何影响项目的最终成败的？为了回答这些问题，Hossain 等[70]建立缺陷管理过程的协作网络，并且研究其中的结构属性对项目表现的影响。结果表明，很多网络指标（例如密度、中心性等）和缺陷管理活动的质量、时间存在相关性。Bhattacharya 等[19]建立缺陷修复过程中的开发者协作关系网和代码提交过程中的开发者协作关系网，并且研究

这些网络的网络指标和缺陷严重程度、模块的维护工作量，以及缺陷数目之间的关系。

“社会－技术一致性”（Socio-Technical Congruence）是通过技术的依赖关系和项目成员的交互程度来衡量团队的协作情况。Kwan 等[96]研究团队的协作情况与软件构建活动的关系，结果发现对于持续构建，高的“社会－技术一致性”会带来更大概率的成功；而对于集成构建，高的“社会－技术一致性”反而会导致成功的概率变小。Cataldo 和 Herbsleb[31]研究“社会－技术一致性”对软件质量和开发生产率的影响。结果发现，协作需要和实际的协作活动之间的差异会大大提高缺陷发生的概率；并且高的“社会－技术一致性”会带来开发者生产率的提升。他们还发现，技术依赖和实际的协作活动之间的一致性不管对于成熟的软件项目，还是对于刚起步的开发环境，都是很重要的。

虽然开发者协作不畅会带来诸如集成失败、重复工作、进度延期、软件缺陷等问题，但是开发者有些时候还是对协作的必要性认识不足。Blincoe 等[22]提出一种方法，能够即时高效地识别和推荐协作。该方法通过捕获集成开发环境中的开发者行为，运用任务的属性和机器学习方法进行协作的精准推荐，从而预防信息过载。该方法在 Mylyn 项目中通过用户访谈和定量的分析进行验证。

3.5.5　基于群体智慧的众包开发

众包属于新兴商业模式，李未院士将众包软件开发称为“群体软件开发”，并提出“群体软件工程”的基本内容，其理论核心是：从封闭走向开放、从精英走向大众、从机械工程到社会工程。其开发原则为“使用者即设计者，使用者即开发者，使用者即维护者”，该开发原则目前主要在 App Store 应用程序中得以体现。目前国内外对于众包领域的研究主要集中在众包概念推广、众包应用模式、众包对社会变革可能带来的影响、众包与外包的联系与区别等理论层面，少部分经验研究主要集中在以 AMT（Amazon Mechanical Turk）为代表的微任务（例如图像识别、创意征集、Logo 设计等）领域。而对复杂任务，特别是众包软件开发的研究较少，已有的研究也大多处于理论层面，缺乏经验及数据支持。

Yang 等[203]基于微任务众包平台 Taskcn 分析了众包接包方的行为模式，并归纳其学习过程与行动策略，为众包任务设计提供了参考建议。Stolee 等[174]针对经验软件研究中数据难以获取的问题，提出通过众包平台征集接包方以收集实验数据，但收集到的数据质量需要进一步被验证。

目前针对众包软件开发方面的研究主要是基于 TopCoder 众包软件开发平台的：Archak[10]将经济学中的“廉价磋商”概念引入众包平台，并探讨荣誉机制与众包软件开发质量之间的关系，该研究表明高等级用户之间存在着“廉价磋商”行为，基于此可自动优化众包社区中的资源配置，并且高等级用户更倾向于提交高质量的软件制品。Mao 等[116]基于众包软件开发历史数据从项目类型、输入质量、输入复杂度、前阶段决策 4 个方面提出共 16 个成本因子，进一步使用多种统计及机器学习方法建立经验定价模型并进行了验证，实验结果表明，所提出的经验定价模型具有较高的准确性，并可以通过回归因子显著性分析抽取出简单易用的定价规则。Li 等[226]分析了影响众包软件开发质量的

因素，并从众包平台以及任务的角度提出了23个质量因子，同时进一步通过回归分析识别出平台任务的平均质量、近期任务数量、文档长度等关键质量因子，并为改善众包软件质量提供了建议。

对于众包群体推荐方面的研究目前也尚处于起步阶段，已知现有的研究是基于AMT平台的微任务众包，Ambati等[7]基于群体兴趣与技能采用隐式建模技术进行群体推荐，并通过实验说明其推荐行为可以提高众包任务的完成质量。Yuen等[206]基于群体过去的任务偏好（例如任务的类别）和表现（例如针对某一类别任务收到的酬金与花费的时间等）提出了一种群体推荐方法，指出该方法可以激励众包群体持续参与众包工作并提升工作质量。

3.5.6 国内研究现状

针对当前研究对于各种bad smells的检测和重构都是独立进行的现状，北京大学的邵维忠等[107]提出了一种检测和消除序列方法，该方法针对不同类型的bad smell，能够简化其检测和消除过程。该方法首先基于示例说明采用检测和消除序列的必要性，并且对于经常发生的bad smell，推荐了检测和消除序列。

对于缺陷修复人推荐问题，中国科学院软件研究所的王青等[200]提出了一种基于主题模型的软件缺陷修复人员推荐方法（DRETOM），该方法利用开发人员参与缺陷修复的历史活动数据对缺陷报告进行主题建模，然后计算开发人员在这些主题上的兴趣度和能力经验值，最终推荐潜在的修复人员。另外，他们[192]还提出一种基于异构网络模型的缺陷报告分派方法（DevNet），用以表征缺陷数据库中开发者之间多种不同类型的协作关系，并在此基础上结合历史缺陷报告修复信息，对缺陷修复人进行推荐。

大连理工大学的江贺等[202]提出基于开发者排序的缺陷报告分派方法，该方法首先根据开发者在缺陷跟踪系统中共同评论缺陷报告的信息构建开发者社会网络，然后根据开发者的贡献对开发者进行排序。在推荐时，首先使用传统的机器学习缺陷报告分派方法给缺陷报告推荐N个修复人，然后根据开发者在开发者社会网络中的顺序重新对N个修复人进行排序。

众包方面，中国科学院软件研究所的王青等[116]基于众包软件开发历史数据从项目类型、输入质量、输入复杂度、前阶段决策4个方面提出共16个成本因子，进一步使用多种统计及机器学习方法建立经验定价模型并进行了验证，实验结果表明，所提出的经验定价模型具有较高的准确性，并可以通过回归因子显著性分析抽取出简单易用的定价规则。另外，他们[226]还分析了影响众包软件开发质量的因素，并从众包平台以及任务的角度提出了23个质量因子，进一步通过回归分析识别出平台任务的平均质量、近期任务数量、文档长度等关键质量因子并为改善众包软件质量提供了建议。

4 国内外研究工作比较和展望

国内在经验软件工程方面的研究起步于21世纪初。一些研究团队，如中国科学院软

件研究所、北京大学、南京大学、清华大学、武汉大学、上海交通大学、复旦大学、北京航空航天大学、西安交通大学、东南大学等开展了卓有成效的工作。CCF 的软件工程专委会于 2013 年成立经验软件工程学组，本次报告的撰写也主要来自该学组的成员。

其中，中科院软件所在成本估算、缺陷预测、需求演化、众包分析以及软件过程和质量改进方面开展了较多国际先进的工作，相关工作在 ICSE、RE、ESEM、ICSM、ICSSP 等会议以及 ASE、ESE、IST 等期刊都有报告；北京大学在基于代码版本库的软件缺陷分析、基于开源代码的微过程挖掘与分析等方面取得了国际领先的研究成果，并发表在 ICSE、FSE、ASE 等软件工程顶级会议上；南京大学在过程改进、质量保障、测试以及缺陷预测等方面的工作上取得很好的进展，相关工作结果在 ICSE、FSE、ESEM、ICSSP 等高水平会议上有报告；复旦大学在基于大规模源码的代码演化分析、代码查找与推荐等方面取得了显著的进展，并将相关结果发表在 ICSE、FSE 等软件工程顶级会议上；武汉大学在需求获取、软件体系结构、缺陷预测以及开源社区分析等方面的工作中也取得了非常好的成果，相关工作结果在 ICSE、FSE、ICSSP 等会议以及 ASE、JSS、IST、TSC 等期刊上都有发表。

CCF 软件工程专委会成立的经验软件工程学组在推动经验软件工程学科的发展，特别是在大学的课程教育方面起到了积极的推动作用。中国科学院软件研究所、北京大学、南京大学还是国际软件工程研究联盟 ISERN（Internation Software Engineering Research Network）的正式会员。正是该联盟组织积极推动了经验软件工程学科的发展，本领域最著名的学者（如 Barry Boehm、Victor Basili、Dieter Rombach 等）都是该联盟的创始成员。

此外，中国的软件产业发展很快，百度、华为等一批企业对经验软件工程的实践需求也开始突显，为我国经验软件工程领域的发展提供了良好的环境和机遇。以上研究团队也都在不同程度上建立了与我国软件企业，特别是龙头企业的合作关系，一些研究成果已经在企业中获得成功应用。

5 结束语

经验软件工程越来越明显地成为软件工程领域主要的研究方法和应用领域，这一结论在近年来的研究论文中得到广泛认可。我国在经验软件工程方面的起步较晚，但机遇和发展空间良好。一些非常好的研究工作也赶上国际先进水平。

总体而言，软件工程也是一门实验科学的认识在最近越来越被大家所认同，特别是在大数据风起云涌的今天，软件的开放性使得在开发、使用、维护和演化各个层面的不确定性日渐突出。而现代质量管理理论认为对数据和信息的逻辑分析和直觉判断是有效决策的基础。经验软件工程从观察、假设、分析到推导结论的方法，为解决开放、不确定性、大数据环境下的软件质量问题提供了有效的方法和途径。但软件工程有其特殊性，主要表现在：

- 稀疏性。大部分的软件工程数据在其关注的属性方面都具有典型的二八性质，譬

如缺陷数据，20% 的模块包含 80% 的缺陷。数据的不均衡性使很多统计方法的应用受到限制。

- 独特性。软件是人类智能的产物，没有任何项目是可以复制的，软件工程的数据也受制于项目、产品、组织的特征，存在严重的数据漂移和语义不统一的问题。
- 可解释性。软件工程数据分析的动机是理解现象产生的原因，支持合理的决策，并开展科学的过程和产品的改进。因此我们进行的实验和分析必须是可解释的，而不能仅限于统计意义。

这些问题是影响经验软件工程研究是否有效可用的关键。此外，对于研究方法，还要在以下几个方面共同努力：

- 经验研究数据。需要更为广泛和丰富的 Benchmark 实验程序。目前 PROMISE、NASA、Ecilips、ISBSG 等提供了一些公共数据集，但面对开放环境，这些还不够。大多数的经验研究都需要“爬取”数据。建立一些公共的开放实验库，对经验软件工程的研究将非常有益。
- 经验分析大多数依然停留在简单的描述统计分析上。这是必要的但还不够，我们应该采用更加系统的统计或机器学习方法，完成精确的分析，使其成为一个严谨的工程学科。经验软件工程学组翻译了一本在国际上被很多大学广泛使用的教材《Experimentation in Software Engineering》，希望在大学相关的课程中就开始训练学生们严谨的经验软件工程方法。
- 经验分析的原型工具和实验数据开发程度远远不够。譬如软件测试需要动态执行程序收集实验数据，这给实验重现带来极大挑战。我们应该提倡研究人员在发表高水平论文的同时，开放原型工具和实验数据，开放的科研成果有助于加强科研合作交流、科研人才培养以及增强科研能力。

此外，随着技术的发展、大数据时代的到来，我们可以从实践中获得海量的数据，但在大数据的环境下，再人工地理解每个项目的应用领域和最佳实践以获得对研究结果的解释将不太可能。要想从这些海量数据中把分散的隐藏的信息挖掘出来，就需要使用数据挖掘相关的技术和算法。与机器学习、社会化网络分析、数据挖掘、自然语言处理、信息检索等学科门技术的结合，可以帮助我们以更丰富的手段和方法获得有价值的信息，并建立有效的理论与预测模型。如何结合软件过程数据的自身特质，选用合适的、在已有的基础上改进的，甚至突破传统的数据挖掘和分析方法，将是非常重要和有意义的。

致谢

本报告在撰写的过程中得到软件工程专委会，尤其是经验软件工程学组许多老师的大力支持，特别是周毓明、彭蓉、王俊杰、王丹丹、杨晨等多位老师和同学，他们都为本文的撰写做出了很多贡献，在这里我们一并表示衷心的感谢！

参考文献

[1] Akiyama F. An Example of Software System Debugging[C/OL]. IFIP Congress (1). 1971, 71: 353-359.

[2] Dunsmore A, Roper M, Wood M. Practical code inspection techniques for object-oriented systems: an experimental comparison[J/OL]. IEEE software, 2003 (4): 21-29. http://www2.dc.ufscar.br/~elis_hernandes/phdproposal/systematicmappings/Q%205.1/5.1%20papers/Dunsmore-Practical%20code%20inspection%20techniques%20for%20object-oriented%20systems%20An%20experimental%20comparison-2003.pdf.

[3] Albayrak Ö, Carver J C. Investigation of individual factors impacting the effectiveness of requirements inspections: a replicated experiment[J/OL]. Empirical Software Engineering, 2014, 19(1): 241-266. https://www.cs.gmu.edu/~offutt/classes/763/papers/Albayrak-reqs-EMSE2014.pdf.

[4] Albrecht A J, Gaffney Jr J E. Software function, source lines of code, and development effort prediction: a software science validation[J/OL]. Software Engineering, IEEE Transactions on, 1983 (6): 639-648.

[5] Aleti A, Buhnova B, Grunske L, et al. Software architecture optimization methods: A systematic literature review[J/OL]. Software Engineering, IEEE Transactions on, 2013, 39(5): 658-683.

[6] Alshayeb M. Empirical investigation of refactoring effect on software quality[J/OL]. Information and software technology, 2009, 51(9): 1319-1326. http://alshayeb.com/publications/J04-Empirical%20Investigation%20of%20Refactoring%20Effect%20on%20Software%20Quality.pdf.

[7] Ambati V, Vogel S, Carbonell J G. Towards Task Recommendation in Micro-Task Markets[C/OL]. Human computation. 2011: 1-4. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.210.5588&rep=rep1&type=pdf.

[8] Amoui M, Salehie M, Tahvildari L. Temporal software change prediction using neural networks[J/OL]. International Journal of Software Engineering and Knowledge Engineering, 2009, 19(07): 995-1014. http://www.stargroup.uwaterloo.ca/~msalehie/papers/IJSEKE09.pdf.

[9] Anvik J, Hiew L, Murphy G C. Who should fix this bug? [C/OL]. Proceedings of the 28th international conference on Software engineering. ACM, 2006: 361-370. http://www.st.cs.uni-sb.de/edu/empirical-se/2006/PDFs/p361.pdf.

[10] Archak N. Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on TopCoder.com[C/OL]. Proceedings of the 19th international conference on World wide web. ACM, 2010: 21-30. http://www.ra.ethz.ch/cdstore/www2010/www/p21.pdf.

[11] Azhar D, Mendes E, Riddle P. A systematic review of web resource estimation[C/OL]. Proceedings of the 8th International Conference on Predictive Models in Software Engineering. ACM, 2012: 49-58. http://www.researchgate.net/profile/Damir_Azhar/publication/236900605_A_Systematic_Review_of_Web_Resource_Estimation/links/0c96051a2a50f49cb0000000.pdf.

[12] Babar M A, Kitchenham B, Jeffery R. Comparing distributed and face-to-face meetings for software architecture evaluation: A controlled experiment[J/OL]. Empirical Software Engineering, 2008, 13(1): 39-62. http://eprints.lancs.ac.uk/64500/.

[13] Banker R D, Kauffman R J, Kumar R. An empirical test of object-based output measurement metrics in a

computer aided software engineering (CASE) environment[J/OL]. Information Systems Working Papers Series, Vol, 1991. http://dl.acm.org/citation.cfm? id=155421.

[14] Bano M, Zowghi D. Users' involvement in requirements engineering and system success[C/OL]. Empirical Requirements Engineering (EmpiRE), 2013 IEEE Third International Workshop on. IEEE, 2013: 24-31. https://opus.lib.uts.edu.au/research/handle/10453/27523.

[15] Bass L. Software architecture in practice, 3rd Edition [M/OL]. Pearson Education India, 2007. http://www.researchgate.net/profile/Rick_Kazman/publication/224001127_Software_Architecture_in_Practice/links/02bfe510fef5da3230000000.pdf.

[16] Bavota G, Oliveto R, Gethers M, et al. Methodbook: Recommending move method refactorings via relational topic models[J/OL]. Software Engineering, IEEE Transactions on, 2014, 40(7): 671-694. https://www.infona.pl/resource/bwmeta1.element.ieee-art-000006684534.

[17] Bayona S, Calvo-Manzano J A, San Feliu T. Review of Critical Success Factors Related to People in Software Process Improvement[M/OL]. Systems, Software and Services Process Improvement. Springer Berlin Heidelberg, 2013: 179-189. https://books.google.com/books? hl=zh-CN&lr=&id=nF25BQAAQBAJ&oi=fnd&pg=PA179&ots=ER7_MijG9m&sig=wezEdxI8xlJ9dRRKTIemiv02zEg.

[18] Bettenburg N, Hassan A E. Studying the impact of social structures on software quality[C/OL]. Program Comprehension (ICPC), 2010 IEEE 18th International Conference on. IEEE, 2010: 124-133. http://www.computer.org/csdl/proceedings/icpc/2010/4113/00/4113a124-abs.html.

[19] Bhattacharya P, Iliofotou M, Neamtiu I, et al. Graph-based analysis and prediction for software evolution [C/OL]. Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012: 419-429. http://ieeexplore.ieee.org/xpl/login.jsp? tp=&arnumber=6227173&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6227173.

[20] Birkhölzer T, Dickmann C, Vaupel J. A Framework for Systematic Evaluation of Process Improvement Priorities[C/OL]. Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on. IEEE, 2011: 294-301. http://www.computer.org/csdl/proceedings/seaa/2011/4488/00/4488a294-abs.html.

[21] Blincoe K, Valetto G, Damian D. Do all task dependencies require coordination? the role of task properties in identifying critical coordination needs in software projects[C/OL]. Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 213-223. https://www.cs.drexel.edu/~kac358/publications/FSE2013-Blincoe.pdf.

[22] Blincoe K, Valetto G, Damian D. Facilitating Coordination between Software Developers: A Study and Techniques for Timely and Efficient Recommendations[J/OL]. IEEE Trans. Softw. Eng. 99, (May 2015), 1-16. http://www.researchgate.net/profile/Giuseppe_Valetto/publication/276206809_Facilitating_Coordination_between_Software_Developers_A_Study_and_Techniques_for_Timely_and_Efficient_Recommendations/links/555281cb08ae980ca606b023.pdf.

[23] Bird C. Community structure in oss projects[R/OL]. Technical report, University of California, Davis, 2006. http://citeseerx.ist.psu.edu/viewdoc/download? doi=10.1.1.122.4758&rep=rep1&type=pdf.

[24] Boehm B W, Madachy R, Steece B. Software cost estimation with Cocomo II with Cdrom[M/OL]. Prentice Hall PTR, 2000. http://www.citeulike.org/group/19011/article/13312192.

[25] Boehm B, Abts C, Chulani S. Software development cost estimation approaches—A survey[J/OL]. Annals of software engineering, 2000, 10(1-4): 177-205. http://www.yamaghani.com/Files/01122012114937.pdf.

[26] Breivold H P, Crnkovic I, Larsson M. A systematic review of software architecture evolution research [J/OL]. Information and Software Technology, 2012, 54(1): 16-40. http://romisatriawahono.net/lecture/rm/survey/software%20engineering/Software%20Architecture/Breivold%20-%20Software%20architecture%20evolution%20research%20-%202012.pdf.

[27] Brunet J, Murphy G C, Serey D, et al. Five years of software architecture checking: A case study of Eclipse[J/OL]. 2014. http://www.dcc.ufmg.br/~mtov/mes/seminarios/9.pdf.

[28] Buckley J, Ali N, English M, et al. Real-Time Reflexion Modelling in architecture reconciliation: A multi case study[J/OL]. Information and Software Technology, 2015, 61: 107-123. http://eprints.brighton.ac.uk/13500/.

[29] Canfora G, Di Penta M, Oliveto R, et al. Who is going to mentor newcomers in open source projects? [C/OL]. Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 2012: 44. http://www.gerardocanfora.net/preprints/whoisgoingtomentornewcomersinopensourceprojectsbygcanforamdipentarolivetoandspanichella/FSE%202012.pdf.

[30] Cao L, Mohan K, Xu P, et al. How extreme does extreme programming have to be? Adapting XP practices to large-scale projects[C/OL]. System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on. IEEE, 2004: 10 pp. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.5437&rep=rep1&type=pdf.

[31] Cataldo M, Herbsleb J D. Coordination breakdowns and their impact on development productivity and software failures[J/OL]. Software Engineering, IEEE Transactions on, 2013, 39(3): 343-360. http://herbsleb.org/web-pubs/pdfs/Cataldo-Coordination-2013.pdf.

[32] Chaumun M A, Kabaili H, Keller R K, et al. A change impact model for changeability assessment in object-oriented software systems[C/OL]. Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on. IEEE, 1999: 130-138. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.1130&rep=rep1&type=pdf.

[33] Sauer C, Jeffery D R, Land L, et al. The effectiveness of software development technical reviews: A behaviorally motivated program of research[J/OL]. Software Engineering, IEEE Transactions on, 2000, 26(1): 1-14. http://www.computer.org/csdl/trans/ts/2000/01/e0001.pdf.

[34] Kemerer C F, Paulk M C. The impact of design and code reviews on software quality: An empirical study based on PSP data[J/OL]. Software Engineering, IEEE Transactions on, 2009, 35(4): 534-550. http://search.proquest.com/docview/195582447?pq-origsite=gscholar.

[35] Denger C, Kolb R. Testing and inspecting reusable product line components: first empirical results [C/OL]. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM, 2006: 184-193. http://www2.dc.ufscar.br/~elis_hernandes/phdproposal/systematicmappings/Q%205.1/5.1%20papers/Denger-Testing%20and%20Inspecting%20Reusable%20Product%20Line-2006.pdf.

[36] Ciolkowski M, Laitenberger O, Biffl S. Software reviews: The state of the practice[J/OL]. IEEE software, 2003(6): 46-51. http://search.proquest.com/docview/215842934?pq-origsite=gscholar.

[37] Shaw M, Clements P. The golden age of software architecture[J/OL]. Software, IEEE, 2006, 23(2): 31-39. http://cat.inist.fr/?aModele=afficheN&cpsidt=17558587.

[38] Condori-Fernandez N, Daneva M, Sikkel K, et al. A systematic mapping study on empirical evaluation of software requirements specifications techniques[C/OL]. Proceedings of the 2009 3rd International Symposium

on Empirical Software Engineering and Measurement. IEEE Computer Society, 2009: 502-505. http://www.researchgate.net/profile/Maya_Daneva/publication/221494786_A_systematic_mapping_study_on_empirical_evaluation_of_software_requirements_specifications_techniques/links/0deec517a94b33f918000000.pdf.

[39] Dang Y, Wu R, Zhang H, et al. ReBucket: a method for clustering duplicate crash reports based on call stack similarity[C/OL]. Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012: 1084-1093. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.2676&rep=rep1&type=pdf.

[40] Denaro G, Margara A, Pezze M, et al. Dynamic Data Flow Testing of Object Oriented Systems[C/OL]. 37th International Conference on Software Engineering, ICSE. 2015, 15. http://www.utdallas.edu/~lxz144130/cs6301-readings/test-generation-denaro-icse15.pdf.

[41] Perry D E, Porter A, Wade M W, et al. Reducing inspection interval in large-scale software development [J/OL]. Software Engineering, IEEE Transactions on, 2002, 28(7): 695-705. http://search.proquest.com/docview/195569606?pq-origsite=gscholar.

[42] Dieste O, Juristo N. Systematic review and aggregation of empirical studies on elicitation techniques [J/OL]. Software Engineering, IEEE Transactions on, 2011, 37(2): 283-304. https://www.infona.pl/resource/bwmeta1.element.ieee-art-000005416730.

[43] Ding W, Liang P, Tang A, et al. How do open source communities document software architecture: An exploratory survey [C/OL]. Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on. IEEE, 2014: 136-145. http://www.computer.org/csdl/proceedings/iceccs/2014/5482/00/5482a136-abs.html.

[44] Ding W, Liang P, Tang A, et al. Causes of Architecture Changes: An Empirical Study through the Communication in OSS Mailing Lists [OL]. http://www.cs.rug.nl/search/uploads/Publications/ding2015cac.pdf.

[45] Du J, Yang Y, Lin Z, et al. A Case Study on Usage of a Software Process Management Tool in China [C/OL]. Software Engineering Conference (APSEC), 2010 17th Asia Pacific. IEEE, 2010: 443-452. http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000005693221.

[46] Díaz J, Pérez J, Garbajosa J. Agile product-line architecting in practice: A case study in smart grids [J/OL]. Information and Software Technology, 2014, 56(7): 727-748. https://www.infona.pl/resource/bwmeta1.element.elsevier-2981a10d-ec2c-3fbf-92f9-111474e9d914.

[47] Etemaadi R, Lind K, Heldal R, et al. Quality-driven optimization of system architecture: Industrial case study on an automotive sub-system[J/OL]. Journal of Systems and Software, 2013, 86(10): 2559-2573. https://www.infona.pl/resource/bwmeta1.element.elsevier-c22b306f-c0f8-3080-9bd8-b6a9174f8a50.

[48] Falessi D, Babar M A, Cantone G, et al. Applying empirical software engineering to software architecture: challenges and lessons learned[J/OL]. Empirical Software Engineering, 2010, 15(3): 250-276. http://www.researchgate.net/profile/Philippe_Kruchten/publication/225678298_Applying_empirical_software_engineering_to_software_architecture_challenges_and_lessons_learned/links/0912f51086a16b9907000000.pdf.

[49] Falessi D, Cantone G, Kazman R, et al. Decision-making techniques for software architecture design: A comparative survey[J/OL]. ACM Computing Surveys (CSUR), 2011, 43(4): 33. http://cat.inist.fr/?aModele=afficheN&cpsidt=25254432.

[50] Feilkas M, Ratiu D, Jürgens E. The loss of architectural knowledge during system evolution: An industrial

case study[C/OL]. Program Comprehension, 2009. ICPC' 09. IEEE 17th International Conference on. IEEE, 2009: 188-197. http://ieeexplore.ieee.org/xpls/abs_all.jsp? arnumber=5090042&tag=1.

[51] Ferrari R, Miller J A, Madhavji N H. A controlled experiment to assess the impact of system architectures on new system requirements[J/OL]. Requirements Engineering, 2010, 15(2): 215-233. http://link.springer.com/article/10.1007/s00766-010-0099-3.

[52] Fischer M, Oberleitner J, Ratzinger J, et al. Mining evolution data of a product family[M/OL]. ACM, 2005. http://dl.acm.org/citation.cfm? id=1083145.

[53] Gao Z, Liang Y, Cohen M B, et al. Making System User Interactive Tests Repeatable: When and What Should we Control? [J/OL]. http://cse.unl.edu/~myra/papers/gao-icse15.pdf.

[54] García F, Piattini M, Ruiz F, et al. FMESP: Framework for the modeling and evaluation of software processes[J/OL]. Journal of Systems Architecture, 2006, 52(11): 627-639. http://www.sciencedirect.com/science/article/pii/S1383762106000658.

[55] Pacheco C. A Web-based Tool for Automatizing the Software Process Improvement Initiatives in Small Software Enterprises[J/OL]. Latin America Transactions, IEEE (Revista IEEE America Latina), 2010, 8(6): 685-694. http://ieeexplore.ieee.org/xpls/abs_all.jsp? arnumber=5688096.

[56] Kalus G, Kuhrmann M. Criteria for software process tailoring: a systematic review[C/OL]. Proceedings of the 2013 International Conference on Software and System Process. ACM, 2013: 171-180. http://dl.acm.org/citation.cfm? id=2486078.

[57] Gharehyazie M, Posnett D, Filkov V. Social activities rival patch submission for prediction of developer initiation in oss projects[C/OL]. Software Maintenance (ICSM), 2013 29th IEEE International Conference on. IEEE, 2013: 340-349. http://ieeexplore.ieee.org/xpls/abs_all.jsp? arnumber=6676905.

[58] Gong L, Lo D, Jiang L, et al. Interactive fault localization leveraging simple user feedback[C/OL]. Software Maintenance (ICSM), 2012 28th IEEE International Conference on. IEEE, 2012: 67-76. http://ieeexplore.ieee.org/xpls/abs_all.jsp? arnumber=6405255.

[59] De Graaf K A, Liang P, Tang A, et al. An exploratory study on ontology engineering for software architecture documentation[J]. Computers in Industry, 2014, 65(7): 1053-1064.

[60] de Graaf K A, Liang P, Tang A, et al. Supporting architecture documentation: a comparison of two ontologies for knowledge retrieval[C]. Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. ACM, 2015: 3.

[61] Halling M, Biffl S, Grünbacher P. An experiment family to investigate the defect detection effect of tool-support for requirements inspection [C]. Software Metrics Symposium, 2003. Proceedings. Ninth International. IEEE, 2003: 278-285.

[62] Halling M, Biffl S, Grünbacher P. An economic approach for improving requirements negotiation models with inspection[J]. Requirements Engineering, 2003, 8(4): 236-247.

[63] Haitzer T, Zdun U. Controlled experiment on the supportive effect of architectural component diagrams for design understanding of novice architects[M]. Software Architecture. Springer Berlin Heidelberg, 2013: 54-71.

[64] Han A R, Jeon S U, Bae D H, et al. Measuring behavioral dependency for improving change-proneness prediction in UML-based design models[J]. Journal of Systems and Software, 2010, 83(2): 222-234.

[65] Hao D, Lan T, Zhang H, et al. Is this a bug or an obsolete test? [M]. ECOOP 2013-Object-Oriented Programming. Springer Berlin Heidelberg, 2013: 602-628.

[66] Harjumaa L, Tervonen I, Vuorio P. Improving software inspection process with patterns[C]. Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on. IEEE, 2004: 118-125.

[67] He M, Zhang H, Yang Y, et al. Understanding the influential factors to development effort in Chinese software industry[M]. Product-Focused Software Process Improvement. Springer Berlin Heidelberg, 2010: 306-320.

[68] van Heesch U, Avgeriou P. Mature architecting- a survey about the reasoning process of professional architects[C]. Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on. IEEE, 2011: 260-269.

[69] van Heesch U, Avgeriou P, Zdun U, et al. The supportive effect of patterns in architecture decision recovery—A controlled experiment[J]. Science of Computer Programming, 2012, 77(5): 551-576.

[70] Hossain L, Zhu D. Social networks and coordination performance of distributed software development teams [J]. The Journal of High Technology Management Research, 2009, 20(1): 52-61.

[71] Howison J, Crowston K. The perils and pitfalls of mining SourceForge[C]. Proceedings of the International Workshop on Mining Software Repositories (MSR 2004). 2004: 7-11.

[72] Huang S K, Liu K. Mining version histories to verify the learning process of legitimate peripheral participants[J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(4): 1-5.

[73] IFPUG F. International Function Point Users Group (IFPUG) Function Point Counting Practices Manual [J/OL]. 2000. http://www. ifpug. org/itips-vtips/.

[74] Isong B, Obeten E. A SYSTEMATIC REVIEW OF THE EMPIRICAL VALIDATION OF OBJECT-ORIENTED METRICS TOWARDS FAULT- PRONENESS PREDICTION[J]. International Journal of Software Engineering and Knowledge Engineering, 2013, 23(10): 1513-1540.

[75] Systems and software engineering: architecture description[S]. ISO, 2011.

[76] Jabangwe R, Börstler J, Šmite D, et al. Empirical evidence on the link between object-oriented measures and external quality attributes: A systematic literature review[J]. Empirical Software Engineering, 2013, 20(3): 640-693.

[77] Javed M, Zdun U. The supportive effect of traceability links in architecture-level software understanding: Two controlled experiments[C]. Software Architecture (WICSA), 2014 IEEE/IFIP Conference on. IEEE, 2014: 215-224.

[78] Jedlitschka A, Pfahl D. Experience- based model- driven improvement management with combined data sources from industry and academia[C]. Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on. IEEE, 2003: 154-161.

[79] Jeffery R, Ruhe M, Wieczorek I. A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data[J]. Information and software technology, 2000, 42(14): 1009-1016.

[80] 贾浩. 软件缺陷分类预测的对比研究[D]. 中国科学院研究生院, 2011.

[81] Maldonado J C, Carver J, Shull F, et al. Perspective-based reading: a replicated experiment focused on individual reviewer effectiveness[J]. Empirical Software Engineering, 2006, 11(1): 119-142.

[82] Jørgensen M, Indahl U, Sjøberg D. Software effort estimation by analogy and "regression toward the mean"[J]. Journal of Systems and Software, 2003, 68(3): 253-262.

[83] Jørgensen M. A review of studies on expert estimation of software development effort[J]. Journal of Systems and Software, 2004, 70(1): 37-60.

[84] Kagdi H, Hammad M, Maletic J. Who can help me with this source code change? [C]. Software Maintenance, 2008. ICSM 2008. IEEE International Conference on. IEEE, 2008: 157-166.

[85] Karner G. Resource estimation for objectory projects[J]. Objective Systems SF AB, 1993, 17.

[86] Kasi B K, Sarma A. Cassandra: Proactive conflict minimization through optimized task scheduling[C]. Software Engineering (ICSE), 2013 35th International Conference on. IEEE, 2013: 732-741.

[87] Kitchenham B A, Taylor N R. Software project development cost estimation[J]. Journal of Systems and Software, 1985, 5(4): 267-278.

[88] Kitchenham B. A procedure for analyzing unbalanced datasets [J]. Software Engineering, IEEE Transactions on, 1998, 24(4): 278-301.

[89] Kitchenham B, Mendes E, Travassos G H. A systematic review of cross-vs. within-company cost estimation studies[C]. Proceedings of the 10th international conference on Evaluation and Assessment in Software Engineering. British Computer Society, 2006: 81-90.

[90] Kitchenham B, Mendes E, Travassos G H. Cross versus within-company cost estimation studies: A systematic review[J]. Software Engineering, IEEE Transactions on, 2007, 33(5): 316-329.

[91] Kim S, Zhang H, Wu R, et al. Dealing with noise in defect prediction [C]. Software Engineering (ICSE), 2011 33rd International Conference on. IEEE, 2011: 481-490.

[92] Klein J, van Vliet H. A systematic review of system-of-systems architecture research[C]. Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures. ACM, 2013: 13-22.

[93] Koru A G, Liu H. Identifying and characterizing change-prone classes in two large-scale open-source products[J]. Journal of Systems and Software, 2007, 80(1): 63-73.

[94] Koziolek H. Sustainability evaluation of software architectures: a systematic review[C]. Proceedings of the joint ACM SIGSOFT conference--QoSA and ACM SIGSOFT symposium--ISARCS on Quality of software architectures--QoSA and architecting critical systems--ISARCS. ACM, 2011: 3-12.

[95] 库燕. 一种基于用例的成本估算方法及工具[D]. 中国科学院研究生院, 2011.

[96] Kwan I, Schröter A, Damian D. Does socio-technical congruence have an effect on software build success? a study of coordination in a software project[J]. Software Engineering, IEEE Transactions on, 2011, 37(3): 307-324.

[97] Lawrence B, Wiegers K, Ebert C. The top risk of requirements engineering[J]. Software, IEEE, 2001, 18(6): 62-63.

[98] Lehrig S, Becker S. Software Architecture Design Assistants Need Controlled Efficiency Experiments: Lessons Learned from a Survey[C]. Proceedings of the 1st International Workshop on Future of Software Architecture Design Assistants. ACM, 2015: 19-24.

[99] Lim S L, Quercia D, Finkelstein A. StakeNet: using social networks to analyse the stakeholders of large-scale software projects[C]. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, 2010: 295-304.

[100] Li W, Henry S. Object-oriented metrics that predict maintainability[J]. Journal of systems and software, 1993, 23(2): 111-122.

[101] 李明树, 何梅, 杨达, 等. 软件成本估算方法及应用 [J]. 软件学报, 2007, 18(4): 775-795.

[102] Li M, Zhang H, Wu R, et al. Sample-based software defect prediction with active and semi-supervised learning[J]. Automated Software Engineering, 2012, 19(2): 201-230.

[103] Li Z, Liang P, Avgeriou P. Application of knowledge-based approaches in software architecture: A

systematic mapping study[J]. Information and Software Technology, 2013, 55(5): 777-794.

[104] Li Z, Liang P, Avgeriou P, et al. An empirical investigation of modularity metrics for indicating architectural technical debt[C]. Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures. ACM, 2014: 119-128.

[105] Li Z, Liang P, Avgeriou P. Architectural Technical Debt Identification based on Architecture Decisions and Change Scenarios[C]. Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), At Montréal, Canada, pp. 211-219.

[106] Liu Q, Mintram R C. Preliminary data analysis methods in software estimation[J]. Software Quality Journal, 2005, 13(1): 91-115.

[107] Liu H, Ma Z, Shao W, et al. Schedule of bad smell detection and resolution: A new way to save effort [J]. Software Engineering, IEEE Transactions on, 2012, 38(1): 220-235.

[108] Liu G, Rong G, Zhang H, et al. The adoption of capture-recapture in software engineering: a systematic literature review[C]. Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. ACM, 2015: 15.

[109] Lu H, Zhou Y, Xu B, et al. The ability of object-oriented metrics to predict change-proneness: a meta-analysis[J]. Empirical software engineering, 2012, 17(3): 200-242.

[110] Lytra I, Gaubatz P, Zdun U. Two controlled experiments on model-based architectural decision making [J]. Information and Software Technology, 2015, 63: 58-75.

[111] López C, Inostroza P, Cysneiros L M, et al. Visualization and comparison of architecture rationale with semantic web technologies[J]. Journal of Systems and Software, 2009, 82(8): 1198-1210.

[112] Ma Y, Luo G, Zeng X, et al. Transfer learning for cross-company software defect prediction[J]. Information and Software Technology, 2012, 54(3): 248-256.

[113] MacDonell S G, Shepperd M J. Comparing Local and Global Software Effort Estimation Models--Reflections on a Systematic Review[C]. Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. IEEE, 2007: 401-409.

[114] Mair C, Shepperd M. The consistency of empirical comparisons of regression and analogy-based software project cost prediction[C]. Empirical Software Engineering, 2005. 2005 International Symposium on. IEEE, 2005: 10 pp.

[115] Malhotra R. A systematic review of machine learning techniques for software fault prediction[J]. Applied Soft Computing, 2015, 27: 504-518.

[116] Mao K, Yang Y, Li M, et al. Pricing crowdsourcing-based software development tasks[C]. Proceedings of the 2013 international conference on Software engineering. IEEE Press, 2013: 1205-1208.

[117] Malavolta I, Lago P, Muccini H, et al. What industry needs from architectural languages: A survey[J]. Software Engineering, IEEE Transactions on, 2013, 39(6): 869-891.

[118] Mattsson A, Lundell B, Lings B, et al. Linking model-driven development and software architecture: a case study[J]. Software Engineering, IEEE Transactions on, 2009, 35(1): 83-93.

[119] Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages[J]. Software Engineering, IEEE Transactions on, 2000, 26(1): 70-93.

[120] 梅宏, 申峻嵘. 软件体系结构研究进展[J]. 软件学报, 2006, 17(6): 1257-1275.

[121] Mendes E, Di Martino S, Ferrucci F, et al. Effort estimation: how valuable is it for a web company to use a cross-company data set, compared to using its own single-company data set? [C]. Proceedings of

the 16th international conference on World Wide Web. ACM, 2007: 963-972.

[122] Mendes E, Di Martino S, Ferrucci F, et al. Cross-company vs. single-company web effort models using the Tukutuku database: An extended study[J]. Journal of Systems and Software, 2008, 81(5): 673-690.

[123] Miao Y, Chen Z, Li S, et al. Identifying Coincidental Correctness for Fault Localization by Clustering Test Cases[C]. SEKE. 2012: 267-272.

[124] Miao Y, Chen Z, Li S, et al. A clustering-based strategy to identify coincidental correctness in fault localization[J]. International Journal of Software Engineering and Knowledge Engineering, 2013, 23(05): 721-741.

[125] Unterkalmsteiner M, Gorschek T, Cheng C K, et al. Evaluation and measurement of software process improvement—a systematic literature review[J]. Software Engineering, IEEE Transactions on, 2012, 38(2): 398-424.

[126] Moløkken K, Jørgensen M. A review of software surveys on software effort estimation[C]. Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on. IEEE, 2003: 223-230.

[127] Moin A H, Khansari M. Bug localization using revision log analysis and open bug repository text categorization[M]. Open Source Software: New Horizons. Springer Berlin Heidelberg, 2010: 188-199.

[128] Moreno-Rivera J M, Navarro E. Evaluation of SPL approaches for WebGIS development: SIGTel, a case study[C]. System Sciences (HICSS), 2011 44th Hawaii International Conference on. IEEE, 2011: 1-10.

[129] Murgia A, Tonelli R, Marchesi M, et al. Refactoring and its relationship with fan-in and fan-out: An empirical study[C]. Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on. IEEE, 2012: 63-72.

[130] Murphy Hill E, Parnin C, Black A P. How we refactor, and how we know it[J]. Software Engineering, IEEE Transactions on, 2012, 38(1): 5-18.

[131] Nakagawa E Y, De Sousa E P M, de Brito Murata K, et al. Software architecture relevance in open source software evolution: a case study[C]. Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International. IEEE, 2008: 1234-1239.

[132] NASA Cost Estimation handbook[OL]. http://www.ceh.nasa.gov/webhelpfiles/Cost_Estimating_Handbook_NASA_2004.htm. 2004.

[133] Engelhart J, Langbroek P. Function Point Analysis (FPA) for Software Enhancement[M]. Jeddah: NESMA, 2001.

[134] Niazi M, Wilson D, Zowghi D. A model for the implementation of software process improvement: A pilot study[C]. Quality Software, 2003. Proceedings. Third International Conference on. IEEE, 2003: 196-203.

[135] Laitenberger O, El Emam K, Harbich T G. An internally replicated quasi-experimental comparison of checklist and perspective based reading of code documents[J]. Software Engineering, IEEE Transactions on, 2001, 27(5): 387-421.

[136] De Oliveira L B R, Felizardo K R, Feitosa D, et al. Reference models and reference architectures based on service-oriented architecture: a systematic review[M]. Heidelberg: Springer, 2010: 360-367.

[137] Palomba F, Bavota G, Di Penta M, et al. Detecting bad smells in source code using change history

information[C]. Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on. IEEE, 2013: 268-278.

[138] Park R E. Software size measurement: A framework for counting source statements[R]. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1992.

[139] Park Y, Park H, Choi H, et al. A study on the application of six sigma tools to PSP/TSP for process improvement[C]. Computer and Information Science, 2006 and 2006 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse. ICIS-COMSAR 2006. 5th IEEE/ACIS International Conference on. IEEE, 2006: 174-179.

[140] Fabrizio Pastore, Leonardo Mariani ZoomIn. Discovering Failures by Detecting Wrong Assertions[C]. Proceedings of International Conference on Software Engineering. 2015: 66-76.

[141] Perry D E, Wolf A L. Foundations for the study of software architecture[J]. ACM SIGSOFT Software Engineering Notes, 1992, 17(4): 40-52.

[142] Peters F, Menzies T, Gong L, et al. Balancing privacy and utility in cross-company defect prediction[J]. Software Engineering, IEEE Transactions on, 2013, 39(8): 1054-1068.

[143] Pfleeger S L, Wu F, Lewis R. Software cost estimation and sizing methods: issues, and guidelines[M]. Colifornia: Rand Corporation, 2005.

[144] Pinzger M, Gall H C. Dynamic analysis of communication and collaboration in OSS projects[M]. Heidelberg: Springer, 2010: 265-284.

[145] Qureshi N, Usman M, Ikram N. Evidence in software architecture, a systematic literature review[C]. Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering. ACM, 2013: 97-106.

[146] Rachatasumrit N, Kim M. An empirical investigation into the impact of refactoring on regression testing[C]. Software Maintenance (ICSM), 2012 28th IEEE International Conference on. IEEE, 2012: 357-366.

[147] Radjenovi? D, Heri? ko M, Torkar R, et al. Software fault prediction metrics: A systematic literature review[J]. Information and Software Technology, 2013, 55(8): 1397-1418.

[148] Rahman F, Khatri S, Barr E T, et al. Comparing static bug finders and statistical prediction[C]. Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 424-434.

[149] Ratzinger J, Sigmund T, Vorburger P, et al. Mining software evolution to predict refactoring[C]. Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. IEEE, 2007: 354-363.

[150] Rong G, Boehm B, Kuhrmann M, et al. Towards context-specific software process selection, tailoring, and composition[C]. Proceedings of the 2014 International Conference on Software and System Process. ACM, 2014: 183-184.

[151] Rost D, Naab M, Lima C, et al. Software architecture documentation for developers: A survey[M]. Heidelberg: Springer, 2013: 72-88.

[152] Saha S K, Selvi M, Buyukcan G, et al. A systematic review on creativity techniques for requirements engineering[C]. Informatics, Electronics & Vision (ICIEV), 2012 International Conference on. IEEE, 2012: 34-39.

[153] Kollanus S, Koskinen J. Survey of software inspection research[J]. The Open Software Engineering Journal, 2009, 3(1): 15-34.

[154] Scacchi W. Free/open source software development: Recent research results and methods[J]. Advances in Computers, 2007, 69: 243-295.

[155] Schultis K B, Elsner C, Lohmann D. Architecture challenges for internal software ecosystems: a large-scale industry case study[C]. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 542-552.

[156] Schultis K B, Elsner C, Lohmann D. Architecture challenges for internal software ecosystems: a large-scale industry case study[C]. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 542-552.

[157] Shahin M, Liang P, Babar M A. A systematic review of software architecture visualization techniques [J]. Journal of Systems and Software, 2014, 94: 161-185.

[158] Shahin M, Liang P, Li Z. Do architectural design decisions improve the understanding of software architecture? two controlled experiments[C]. Proceedings of the 22nd International Conference on Program Comprehension. ACM, 2014: 3-13.

[159] Shaw M, Clements P. The golden age of software architecture[J]. Software, IEEE, 2006, 23(2): 31-39.

[160] Sharafat A R, Tahvildari L. Change prediction in object- oriented software systems: A probabilistic approach[J]. Journal of Software, 2008, 3(5): 26-39.

[161] Shi Q, Chen Z, Fang C, et al. Measuring the Diversity of a Test Set With Distance Entropy[J/OL]. IEEE TRANSACTIONS ON RELIABILITY, 2015. http://ieeexplore. ieee. org/stamp/stamp. jsp? arnumber = 7116633&tag = 1.

[162] Sivashankar M, Kalpana A M. A framework approach using CMMI for SPI to Indian SME'S[C]. Innovative Computing Technologies (ICICT), 2010 International Conference on. IEEE, 2010: 1-5.

[163] De Silva L, Balasubramaniam D. Controlling software architecture erosion: A survey[J]. Journal of Systems and Software, 2012, 85(1): 132-151.

[164] SIR[OL]. http://sir. unl. edu/php/previewfiles. php.

[165] Sjøberg D I K, Hannay J E, Hansen O, et al. A survey of controlled experiments in software engineering [J]. Software Engineering, IEEE Transactions on, 2005, 31(9): 733-753.

[166] Soh Z, Khomh F, Guéhéneuc Y G, et al. Towards understanding how developers spend their effort during maintenance activities[C]. Reverse Engineering (WCRE), 2013 20th Working Conference on. IEEE, 2013: 152-161.

[167] Qinbao Song, Martin J. Shepperd, Carolyn Mair: Using Grey Relational Analysis to Predict Software Effort with Small Data Sets[J]. IEEE METRICS 2005: 35.

[168] Qinbao Song, Martin J. Shepperd, Michelle Cartwright, Carolyn Mair: Software Defect Association Mining and Defect Correction Effort Prediction[J]. IEEE Transactions on Software Engineering. 2006: 32 (2): 69-82.

[169] Stacy K L, Nicholas A K, Letha H E. Source Code Retrieval for Bug Localization using Latent Dirichlet Allocation[C]. Proceedings of 15th Working Conference on Reverse Engineering, Antwerp, Belgium, 2008: 155-164.

[170] Mark Staples, Mahmood Niazi. Two case studies on small enterprise motivation and readiness for CMMI [C]. Proceedings International Conference on Product Focused Software , pp. 63-66, ACM (2010).

[171] Stefan Biffl, et al. Investigating the Defect Detection Effectiveness and Cost Benefit of Nominal Inspection

Teams[J]. IEEE Transactions on Software Engineering . 2003.

[172] Stefan Biffl, et al. A family of experiments to investigate the effects of groupware for software inspection [J]. Automated Software Engineering. 2006.

[173] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, David Redmiles. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects[C]. In Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, 2015: 1379-1392.

[174] Stolee K T, S Elbaum. Exploring the use of crowdsourcing to support empirical studies in software engineering [C]. Proceedings International Symposium on Empirical Software Engineering and Measurement. 2010.

[175] Ting Su, Zhoulai Fu, Geguang Pu, Jifeng He, Zhendong Su. Combining Symbolic Execution and Model Checking for Data Flow Testing[C]. Proceedings of International Conference on Software Engineering. 2015: 654-665.

[176] Muhammad Sulayman, Emilia Mendes. An extended systematic review of software process improvement in small and medium Web companies[C]. Proceeding Internation Conference of Evaluation & Assessment in? Software? Engineering, 2011: 134-143.

[177] R B Svensson, M Host, B Regnell. Managing quality requirements: A systematic review[C]. Proceedings of 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2006: 261-268.

[178] Hee Beng Kuan Tan, Yuan Zhao, Hongyu Zhang. Estimating LOC for information systems from their conceptual data models[C]. Proceedings of International Conference on Software Engineering. 2006: 321-330.

[179] Hee Beng Kuan Tan, Yuan Zhao, Hongyu Zhang. Conceptual data model-based software size estimation for information systems[J]. ACM Transactions on Software Engineering Methodology. 2009: 19(2): 1-37.

[180] Taneja K, D Dig, T Xie. Automated detection of API refactorings in libraries[C]. in Proceedings of IEEE/ACM international conference on Automated software engineering. 2007.

[181] A Tang, M A Babar, L Gorton, J Han. A survey of architecture design rationale[J]. Journal of systems and software, 2006: 79(12): 1792-1804.

[182] Valerio Terragni, Shing-Chi Cheung, Charles Zhang. RECONTEST: Effective Regression Testing of Concurrent Programs [C]. Proceedings of International Conference on Software Engineering. 2015: 246-256.

[183] D Tofan, M Galster, P Avgeriou. Difficulty of architectural decisions-a survey with professional architects [C]. Proceedings of the 7th European Conference on Software Architecture (ECSA), Montpellier, France, 2013: 192-199.

[184] D Tofan, M Galster, P Avgeriou, W Schuitema. Past and future of software architectural decisions - A systematic mapping study[J]. Information and Software Technology, 2014: 56(8): 850-872.

[185] N Tsantalis, A Chatzigeorgiou, G Stephanides. Predicting the probability of change in object-oriented systems [J]. IEEE Transactions on Software Engineering, 2005, 31(7): 601-614.

[186] Jingxuan Tu, Lin Chen, Yuming Zhou, Jianjun Zhao, Baowen Xu. Leveraging Method Call Anomalies to Improve the Effectiveness of Spectrum-Based Fault Localization Techniques for Object-Oriented Programs [C]. QSIC 2012: 1-8.

[187] UKSMA. MK II Function Point Analysis: Counting Practices Manual, Version 1.3.1[C/OL]. United Kingdom Software Metrics Association (UKSMA), 1998. http://www.uksma.co.uk/public/ mkIIr131.pdf.

[188] F Vogelezang. COSMIC full function points the next generation of functional sizing[C/OL]. In: Software Measurement European Forum SMEF 2005, 2005.

[189] L Votta, et al. Does every inspection need a meeting? [J/OL]. ACM Software Engineering Notes. 1993.

[190] F Walkerden, R Jeffery. An Empirical Study of Analogy-based Software Effort Estimation[J]. Journal of Empirical Software Engineering, 1999: 4(2): 135-158.

[191] Yong Wang, Qinbao Song, Stephen G. MacDonell, Martin J. Shepperd, Junyi Shen. Integrate the GM(1, 1) and Verhulst Models to Predict Software Stage Effort[J]. IEEE Transactions on Systems, Man, and Cybernetics, 2009: Part C. 39(6): 647-658.

[192] Wang S, Zhang W, Yang Y, Wang Q DevNet. Exploring Developer Collaboration in Heterogeneous Network of Bug Repositories[C]. Proceedings ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'13), Oct 2013, Maryland, USA. 2013: 193-202.

[193] J Wen, S Li, Z Lin. Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models[J]. Information and Software Technology, 54 (1): 41-59, 2012.

[194] Michael W Whalen, Suzette Person, Neha Rungta, Matt Staats, Daniela Grijincu . A Flexible and Non-intrusive Approach for Computing Complex Structural Coverage Metrics[C]. Proceedings of International Conference on Software Engineering. 2015: 506-516.

[195] I. Wieczorek, M Ruhe. How Valuable is company-specific Data Compared to multi-company Data for Software Cost Estimation? [C]. Proceedings of the 8th IEEE International Symposium on Software Metrics, 2002: 237-246.

[196] B J Williams, J C Carver. Characterizing software architecture changes: A systematic review[J]. Information and Software Technology, 2010: 52(1): 31-51.

[197] Chu-Pan Wong, Yingfei Xiong, Hongyu Zhang, Dan Hao, Lu Zhang, Hong Mei. Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis[C]. Proceedings International Conference on Software Maintenance and Evolution. 2014: 181-190.

[198] Edmund Wong, Lei Zhang, Song Wang, Taiyue Liu, Lin Tan. DASE: Document-Assisted Symbolic Execution for Improving Automated Software Testing[C]. Proceedings of International Conference on Software Engineering. 2015: 620-631.

[199] 伍书剑. 基于系统动力学技术的缺陷预测模型及经验研究[D]. 中国科学院研究生院, 2010.

[200] Xie X, Zhang W, Yang Y, Wang Q. DRETOM: developer recommendation based on topic models for bug resolution[C]. Proceedings of the 8th International Conference on Predictive Models in Software Engineering (PROMISE'12). September 21-22, 2012, Lund, Sweden. ACM, 19-28.

[201] Xu J, G Madey. Exploration of the open source software community[C]. Proceedings of North American Association for Computational Social and Organizational Science (NAACSOS). 2004. Pittsburgh, PA, USA.

[202] Xuan J, Jiang H, Ren Z, Zou W. Developer Prioritization in Bug Repositories[C]. Proceedings of 34th International Conference on Software Engineering. June. 2012, 25-35.

[203] Yang J, L A Adamic, M S Ackerman. Crowdsourcing and knowledge sharing: strategic user behavior on taskcn[C]. Proceedings of the 9th ACM conference on Electronic commerce (EC'08). New York,

NY, USA.

[204] Yu L, S, Ramaswamy. Mining cvs repositories to understand open-source project developer roles[C]. Proceedings of the 4th International Workshop on Mining Software Repositories. 2007: 8.

[205] Yuan Z, Yu L, Liu C. Bug prediction method for fine-grained source code changes[J/OL]. Ruan Jian Xue Bao/Journal of Software, 2014, 25(11): 2499-2517 (in chinese). http://www.jos.org.cn/1000-9825/4559.htm. [doi: 10.13328/j.cnki.jos.004559].

[206] Yuen M C, I King, K S Leung. Task Matching in Crowdsourcing[C]. Proceedings of the 4th International Conference on Cyber, Physical and Social Computing. 2011: 409-412.

[207] Zhang W, Yang Y, Wang Q. Network analysis of OSS evolution: an empirical study on ArgoUML project [C]. In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, Szeged, Hungary, 2011: 71-80.

[208] Zhang, He, Muhammad Ali Babar, Paolo Tell. Identifying relevant studies in software engineering[J]. Information and Software Technology 53.6 (2011): 625-637.

[209] Hongyu Zhang, Liang Gong, Steven Versteeg: Predicting bug-fixing time: an empirical study of commercial software projects[C]. Proceedings of International Conference on Software Engineering. 2013: 1042-1051.

[210] Yuming Zhou, Hareton Leung. Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults[J]. IEEE Transactions on Software Engineering. 2006: 32(10): 771-789.

[211] Zhou Y, Leung H. Predicting object-oriented software maintainability using multivariate adaptive regression splines[J]. Journal of Systems and Software, 2007: 80(8): 1349-1361.

[212] Zhou Y, Leung H K, N Xu B. Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness[J]. IEEE Transactions on Software Engineering. 2009: 35(5): 607-623.

[213] Zhou J, Zhang H, David L. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports [C]. Proceedings of 34th International Conference on Software Engineering, Zurich, Switzerland, 2012: 14-24.

[214] Zhu X, Song Q, Sun Z. Automated Identification of Change-Prone Classes in Open Source Software Projects[J]. JSW, 2013: 8(2): 361-366.

[215] Zimmermann T, et al. Mining Version Histories to Guide Software Changes[C]. Proceedings of 26th International Conference on Software Engineering. 2004: 429-445.

[216] Zimmermann T, A Zller, P Weigerber, et al. Minin version histories to guide software changes[J]. IEEE Transactions on software Engineering, 2005, 31(6): 429-445.

[217] Zimmermann T, Nagappan N. Predicting Defects using Network Analysis on Dependency Graphs[C]. Proceedings of the 30th International Conference on Software Engineering, 2008: 531-540.

[218] Zimmermann T, Nagappan N, Gall H. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process[C]. Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, 2009: 91-100.

[219] Kitchenham B, Dybå T, Jørgensen M. Evidence-based Software Engineering [C/OL]. Proceedings of 26th International Conference on Software Engineering (ICSE), 2004: pp. 273-281.

[220] Kitchenham B. Guidelines for Performing Systematic Literature Reviews in Software Engineering [R]. Software Engineering Group, School of Computer Science and Mathematics, Keele University, and

Department of Computer Science, University of Durham (2007).

[221] Kitchenham B. Procedures for Undertaking Systematic Reviews. Computer Science Department [R], Keele University and National ICT Australia (2004).

[222] Hossain E, Babar M. A, Paik H. Using Scrum in Global Software Development: A Systematic Literatur Review [C]. Proceedings of the 4th International Conference on Global Software Engineering. IEEE Press, Los Alamitos, pp. 195-204.

[223] Beijun Shen, Ruan Tong. A Case Study of Software Process Improvement in a Chinese Small Company [C]. Proceedings of International Conference on Computer Science and Software Engineering (CSSE 2008). Dec. 2008, Wuhan, China, PP. 609-612.

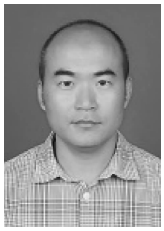
[224] E Guzman, W Maalej. How do users like this feature? A finegrained sentiment analysis of App review [C]. Proc. 22nd IEEE InternationalRequirements Engineering Conference (RE2014), Karlskrona, Sweden, August 2014, pp. 153-162.

[225] N Chen, J Lin, S Hoi, X Xiao, B Zhang. AR-miner: mininginformative reviews for developers from mobile App marketplace [C]. Proceedings of 36th International Conference on Software Engineering (ICSE2014), Hyderabad, India, Jun. 2014, pp. 767-778.

[226] Ke Li, Junchao Xiao, Yongji Wang, Qing Wang. Analysis of the Key Factors for Software Quality in Crowdsourcing Development: An Empirical Study on TopCoder. com [C]. Proceedings of the 37th Annual International Computer Software & Applications Conference (Compsac2013), July 22-26, 2013.

作者简介

王 青 女，博士，研究员、博士生导师，现任中国科学软件研究所副总工程师，主要研究方向包括：软件过程、软件质量保障、需求工程、知识工程等。目前主要的社会兼职为中国电子学会云计算专委会委员，全国信息技术标准化委员会软件质量测试工作组（SAC/TC28/SC7/WG1）副组长，以及 CMMI 认证授权的主任评估师。她主持和承担了多项国家重点/重大项目和国内外重大合作项目，获得国家及省部委科技进步奖 10 余次。还曾应邀担任多个国际会议的程序委员会主席或委员，同时担任 ESEIW/ESEM 2015（ACM 国际经验软件工程与度量会议）大会主席，也是 IST、JSS 等国际期刊审稿人，近年有 5 本论/编著、100 余篇论文在国际、国内重要学术刊物以及国际会议上发表。也曾在中国科学院大学主讲《高级软件工程》课程。详细履历见 http://itechs. iscas. ac. cn/cn/education/wq. htm。



张 贺 哲学博士（Ph. D），南京大学软件学院教授，博士生导师，国际软件工程研究联盟（ISERN）成员、南京大学代表，中国计算机学会高级会员、软件工程专委会委员。毕业于澳大利亚新南威尔士大学（UNSW），以最优成绩获博士学位（Ph. D）。师从世界顶级软件工程专家（澳）Ross Jeffery 教授和（英）Barbara Kitchenham 教授。在欧洲和澳洲从事软件工程研究与实践 10 余年。先后在爱尔兰国家软件工程研究中心

（Lero）任研究员，在澳大利亚国家信息与通信科学院（NICTA）任资深研究员，并在（爱）利默里克大学（UL）信息技术系和（澳）新南威尔士大学（UNSW）计算机科学与工程系任特聘讲师。也曾负责主持多个爱尔兰、澳大利亚、中国等国家级科研基金项目。近年来，著有英文专著一部，并在国际软件工程大会（ICSE）和 Empirical Software Engineering、Information and Software Technology、Journal of Systems and Software、IEEE Transactions on Service Computing、Journal of Software：Evolution and Process 等国际主要软件工程期刊和会议上发表论文 90 余篇（约 40% 为第一作者），其中 8 篇获最佳论文奖。2012 年入选“登峰计划”（A-），2013 年起，任教于南京大学。

张　莉　北京航空航天大学计算机学院/软件学院教授，博士生导师，中国计算机学会软件工程专委会委员及教育专委副主任委员，教育部软件工程专业教学指导委员会委员，全国工程专业学位研究生教育指导委员会软件工程领域协作组组长。她的研究兴趣是：软件工程（系统设计、软件重用、领域/系统建模、经验软件工程等）。

梁　鹏　武汉大学软件工程国家重点实验室教授、博士生导师，主要研究方向为软件体系结构、知识驱动的软件工程。

周明辉　北京大学信息科学技术学院副教授，主要研究兴趣是：通过挖掘软件开发支持工具（version control system、issue tracking system、mailing list 等）记录的历史数据，研究软件产品和程序员及其工作文化之间的关系（尤其是开源项目）。在软件工程领域顶级国际期刊及 TSE、ICSE 和 FSE 会议等发表 40 多篇论文，获 FSE2010 的 ACM SIGSOFT 杰出论文奖和 COMPSAC 2012 最佳论文奖。还多次担任国际会议 PC，如 FSE 2014 Research Demo PC co-chair 和 Internetware 2014 PC co-chair 等。