软件工程进展

CCF 软件工程专业委员会

摘　要

本报告首先简要回顾了软件工程的研究目标与基本内容，然后比较详细地介绍了软件工程近期的热点与进展，以及国内学者在软件工程方面所取得的主要成果。报告还根据软件的特点分析了软件工程面临的挑战，并进行了展望。

关键词：软件工程，软件开发，软件质量

Abstract

This report firstly reviewed research goal and content of software engineering. Secondly, the report introduced the hot topics and new research results in recent years, and main software engineering research results of mainland researchers. Challenges and future research directions are analyzed in the end.

Keywords: software engineering, software development, software quality

1　引言

软件是人类智力活动的产物，而且其形态与运行环境密切相关。软件开发是一个充满“创新”活动的过程，早期有种观点甚至将程序设计看做一种“艺术”。既然如此，为什么要强调软件开发中的“工程”呢？众所周知，软件给我们的工作、学习、生活带来的好处巨大，因此社会对软件产品的需求非常迫切，从“艺术”角度看到的程序与社会需要的“产品”往往还有很大的差距。如果从“产品”的角度看软件的话，软件开发的“质量”、“进度”、“预算”控制都非常困难，这就是“软件危机”。“软件危机”自1968 年提出之后的40 多年来，一直在困扰着相关研究人员。而在工程管理领域，人们经过数百年的探索，总结了一系列的原则与方法，有效地控制了许多项目的“质量”、“进度”、“预算”等问题。因此，一个自然的想法是在软件开发过程中引入工程管理的原则与方法，来应对“软件危机”。“软件工程”一词也因此在“软件危机”出现的同时被提出。IEEE 在1993 年将“软件工程”定义为“将系统化的、严格约束的、可量化的方法应用于软件的开发、运行和维护，即将工程化应用于软件”。

2　软件工程的研究目标

“软件工程”术语第一次出现在1968 年的 NATO 会议上。这次会议以及而后的一些

会议集中讨论了大型软件开发项目中出现的软件质量与开发效率等问题，以着手解决软件开发失控的所谓“软件危机”。如何提高软件质量与开发效率从此一直成为软件工程的主要研究目标。

2.1 软件质量

软件质量是软件产品和服务能够表明满足客户要求的全部性质和特征。软件质量可以通过一组属性予以度量，这组属性是软件质量高低的表征。

由于软件的特殊性，人们对于软件质量的认识至今尚不完善。IEEE 在 1991 年提出的 ISO9126 定义的六类软件质量模型是其中一个公认的重要参考。目前比较受关注的 5 种相对并列的软件质量属性分别为：正确性、健壮性、安全性、高效性、易维护性[1]。

1）“正确性”（Correctness），主要指程序的运行结果是否与预期值相同，如果不相同，则视为结果不正确，该程序包含一个错误。正确性的考虑比较单纯，主要是从输入到输出这个函数映射的角度看待计算过程，特别关系输出结果的正确与否，而不考虑输入数据的具体来源与错误输出结果的后果如何。与正确性相对的是健壮性与安全性：正确性考虑的是软件是否按照预先的设定执行，健壮性与安全性则考虑软件是否在一定条件下执行设定之外的事情。

2）“健壮性”（Robustness），主要是指系统中是否能控制软件内外客观不良事件的发生，以保持系统的基本服务质量。例如，是否会发生因死锁而导致系统不工作、是否因为内存泄漏而导致系统崩溃、是否因产生数据竞争而导致系统出现错误状态等。健壮性与正确性之间存在一定的关联：首先，不正确的输出有可能影响包含软件的系统整体上的健壮性；其次，因系统崩溃等原因导致得不到结果有时被认为是“不正确”的一种特殊表现。

3）“安全性”（Security），主要指软件是否存在安全漏洞，是否可能由于人为攻击，造成信息泄露（保密性攻击）、数据损失（完整性攻击）或者系统不能正常工作（可用性攻击）等不良后果。安全性与健壮性之间的主要区别在于：导致系统不安全的因素是人为因素，而导致系统不健壮的因素是软件本身。另外，二者之间也存在关联：许多系统因为健壮性存在缺陷（例如，字符串溢出）而导致安全性缺陷（恶意注入）。

4）“高效性”（Efficiency），包括时间高效性与空间高效性。时间高效性主要是指系统响应时间小、吞吐率高，空间高效性主要是占用资源（例如，CPU、内存、数据库连接、网络连接等）少。类似内容还包括：同时服务的客户数目等。

5）“易维护性”（Maintenance），主要指系统是否易于理解、易于根据需求的变化对系统进行调整。

由于软件在逻辑上的复杂性，软件质量每一个属性的度量与保障都不容易，其中重要的原因包括：①受有限的时间和资金资源的限制；②一些质量特征与其他特征之间是相互冲突的，例如，对一个特定的环境，一个系统很难既十分方便又具有最大功效。近年来，随着计算机网络逐步成为主要的软件运行环境，软件属性的保障面临着越来越多

的挑战。因此，人们在质量的基础上更多地开始关注软件的可信性，更加强调用户对软件制品各个属性的预期。

2.2　软件开发效率

软件质量在本质上是由软件开发过程决定的。工程项目，尤其是项目质量，通常会受到成本（通常是时间和资源）的严格约束。对于软件工程而言，由于最终的产品是逻辑产物，硬件资源的开销几乎可以忽略，需要的资源主要是开发人员。因此，所谓软件开发效率，即是指用多少开发人员在多少时间内开发出预期的软件。

1）开发人员。包含数量与质量两个方面。对于一个软件项目，人们往往首先关注投入了多少人员，然后才注意所投入人员的具体素质。实际上，在软件工程领域，开发人员的素质对于项目的成败非常重要。不仅如此，有统计表明，高水平的开发人员效率常常是一般人员的数十倍多。软件工程人员素质的提高是软件工程教育的核心问题，软件工程教育因此是软件工程十分重要的内容。受一些因素限制，本文暂不展开在这方面的讨论。

2）开发时间。从项目启动开始，到软件正式上线的时间，往往是最受关注的时间段。人们总是希望尽快得到所期望的软件，并往往希望通过增加人员缩短时间。但实际上，由于不同开发任务之间存在大量的很强的关联性，导致许多任务无法并发进行。因此，“向进度落后的项目中增加人手，只会使进度更加落后”[2]。这就导致了，尽管软件的开发成本主要由人员（例如，多少人）与时间（例如，多少月）决定，但开发成本不宜用“人月”计算。

在实际的软件项目中，开发过程的量化往往比质量的度量更加困难。这主要是由于不同软件项目的差异往往比较大。这是软件工程项目与传统的工程项目相比非常特殊的地方：与软件工程项目相比，传统的许多工程项目，例如土木工程、机械工程等，项目之间的相似度大一些。尤其是，对于传统的工程项目，占项目开销主体的通常是实施环节。实际上，软件的开发过程更接近于传统项目的设计环节，软件工程项目的“部署”环节与传统工程项目的实施环节相近。而部署环节在软件工程项目中的开销比重往往很小。

3　软件工程基本内容

对于一个具体的软件工程项目，软件工程人员需要在对项目进行总体分析后，选择合适的开发过程模型、合适的开发方法、合适的质量度量方法以及具体的开发环境。从研究角度看，软件工程研究主要是探索针对不同软项目的更好的开发过程模型、开发方法、质量度量方法与更好的开发环境[3]。

3.1 开发过程模型

软件开发过程是将用户需求转换为软件系统（产品）的一组有序的活动。其中，每一活动是由一组任务定义的，而任务是把输入转换为输出的一组操作。尽管在实践中，每项软件工程都有自己特定的软件开发过程，但主要的软件开发模型包括：瀑布模型、迭代模型、螺旋模型以及第四代模型等。瀑布模型是其他模型的基础，将软件开发的分析、设计、编码、测试等活动规定为依固定顺序联接的若干阶段工作，形如瀑布流水，最终得到预期的软件产品。瀑布模型将大型软件的开发活动进行有机分解，为开发机构有效地组织实施，制订开发规范，控制软件质量创造了有利的条件，是许多后续开发模型的基础。对于不同的目标软件系统，需要开发人员分析其特点，选择适合的开发模型。

3.2 开发方法

软件开发方法是指软件开发过程所遵循的办法和步骤。典型的软件开发方法有：结构化方法、面向数据结构方法、面向对象方法、构件化开发方法等。结构化方法是一种系统化的软件系统建模方法，包括结构化分析和结构化设计。面向数据结构方法是结构化方法的变形，其中，着重数据结构而不是数据流。面向对象方法是一种以对象、对象关系等来构作软件系统模型的系统化方法。包括面向对象分析和设计。其中，对象由封装的属性和操作组成；对象类是具有相似属性、操作和关系的一组对象的描述。类是系统建模、系统设计以及实现等活动可复用的基础。构件化开发方法的核心是可复用的软件构件。构件是比对象粒度更大的软件实体，通常是由第三方组织开发的，其实现往往与分布对象技术有很大的关联。

3.3 软件度量

软件度量的核心是软件质量度量。对软件质量的度量主要依靠对软件的分析，并尽量发现其中存在的缺陷，以消除缺陷，提高质量。从分析方法角度，可以分为静态分析与动态分析，静态分析是指在不运行软件前提下的分析过程。静态分析的对象一般是程序源代码，也可以是目标码（例如 Java 的 byte code），甚至可以是设计模型等形态的制品。动态分析是通过运行具体程序并获取程序的输出或者内部状态等信息来验证或者发现软件性质的过程。与静态分析相比，动态分析需要运行系统，因此通常要向系统输入具体的数据。另外，由于有具体的数据，因此动态分析结果更精确，但同时只是对于特定输入情况精确，对于其他输入的情况则不能保证。从分析对象角度，又可以分为对软件生命周期不同阶段制品的分析，包括对需求分析结果、设计结果、代码模块、总体系统、上线软件等[4]。

3.4　开发工具与环境

纵观软件工程40多年来的进展，除了一些原则性的指导方法外，可用的计算机辅助软件工程（Computer Aided Software Engineering，CASE）工具和环境的出现是软件工程实践进步的最好体现。CASE工具大体上可分为两类：一类是支持工程管理活动的工具，例如，配置管理工具，成本估算工具，调度工具以及文档工具等；另一类是支持开发活动的工具，包括设计工具（如原型速成工具，建模工具等），程序设计辅助工具（如调试工具，代码生成器等），测试工具以及维护工具。CASE环境是CASE工具的集成，支持整个软件开发工作。

集成开发环境（Integrated Developing Environment，IDE，例如，微软的Visual Studio、IBM的Eclipse等）、建模工具（例如，支持统一建模语言UML的IBM Rational Rose等）、配置管理工具（如CVS、SVN、Git等）、缺陷跟踪系统（例如Issue Tracker）以及测试工具（如Junit等）在大量软件工程项目中的广泛应用即是对CASE所取得成功的最有力说明。

4　近期热点与进展

4.1　软件开发过程

总的来看，近期人们对适应各类软件的基本开发方法的关注有所下降。而面向某类特定软件的方法相对得到了很多的关注与应用。这里选取敏捷方法、模型驱动的方法、面向服务的开发方法等作为代表性进展方法进行介绍。

1）敏捷（Agile）方法。这是一种从1990年代开始逐渐引起广泛关注的新型软件开发方法，主要特征是能适应快速变化的需求[5,6]。相对于“非敏捷”，该方法更强调程序员团队与业务专家之间的紧密协作、面对面的沟通（认为比书面的文档更有效）、频繁交付新的软件版本、紧凑而自我组织型的团队、能够很好地适应需求变化的代码编写和团队组织方法，也更注重软件开发中人的作用。与传统的瀑布式开发相比，瀑布模型适合需求明确、工期严格的大型关键软件，而敏捷方法适合需求容易变化、开发周期不长的小型软件。当应用于规模较小、质量要求不那么高的软件时，瀑布模型的主要问题是它的严格分级导致的自由度降低，项目早期做出的承诺可能导致对后期需求的变化难以调整，代价高昂。敏捷方法则在几周或者几个月的时间内完成相对较小的功能，强调的是能将尽早将尽量小的可用的功能交付使用，并在整个项目周期中持续改善和增强。

2）模型驱动的开发方法。这主要是由对象管理组织（Object Management Group，OMG）自2001开始倡导的一种方法，有时也称为模型驱动的软件体系结构（Model Driv-

en Architecture，MDA)[7]。MDA 方法的核心是基于适当的特定于领域的语言（Domain-Specific Language，DSL)，通过独立于平台的模型（Platform Independent Model，PIM）来定义系统功能。然后，在一个给定的平台上（如 CORBA、.NET、Web Service 等）PIM 被转换为一个或者多个特定于平台的模型（Platform-Specific Models，PSM)，并继续利用模型生成尽可能多的代码，从而提高软件开发的自动化程度。MDA 方法特别关注从抽象的、人的想法为核心的模型图逐步向具体的代码为核心的制品的自动映射。因此，模型的自动（半自动）转换是 MDA 方法的重要内容。由于 MDA 方法中模型之间的追踪性比传统方法中文档的追逐性要好，因此该方法对于经常需要演化软件的支持具有优势。目前取得的成果主要体现在：模型的标准化，模型之间的追踪，模型的自动转换等。

3）面向服务的开发方法。该方法是面向过程的方法（结构化方法）、面向对象方法、构件化开发方法的延续。过程（Procedure）最基本的软件模块，是大型程序从混沌向结构化走出的第一步，每个过程都有具体的调用格式（在 C/C ++ 中，该格式用头文件来说明)；对象（Object）是方法与数据的封装体，是类（Class）的实例，其中的方法与过程有直接的对应关系。构件（Component）是更大粒度的构造模块，通常由一个或多个类组成的实体，也可以直接由一个或多个过程组成。服务（Service）是随着计算机网络的发展逐渐形成的新软件形态（Software as a Service，SaaS)。服务强调的是软件实体的外在表现，其内在实现则仍然是由某个构件、某个对象，甚至某个过程完成。不同构造模块出现的时间不同，后者直接发展了前者，后者的实现中往往包含了前者，只是强调的重点有所改变，但后者不是替代了前者。目前取得的主要成果主要包括：服务的开发，服务的查找，服务的测试，服务的组装、服务的管理等[8]。尽管面向服务的方法至今尚未完全成型，但可以相信，在云计算（SaaS + PaaS + IaaS）的推动下，在学术界与企业界的双重努力下，这类方法会逐渐成熟并被广泛应用。

4.2 软件库复用

复用是提高软件开发效率与软件质量的有效途径。对象化、构件化、服务化等诸多开发方法中都隐式地包含了复用的思想。

软件库/框架作为显式复用的主要载体，在实际软件开发中得到了越来越广泛的使用。随着软件库在数量上和规模上的不断增大，如何方便程序员正确使用软件库也逐渐成为了需要关注的问题。近十几年来，研究人员针对这一问题展开了较为系统的研究，能够针对程序员的需要挖掘和推荐软件库 API 的使用规则。具体而言，现有的研究可以从使用软件库的客户代码、软件库的实现代码，甚至是软件库的文档和注释里挖掘软件库 API 的使用规则，而挖掘出来的使用规则既可以是简单的调用顺序，也可以是复杂的调用子图。在此基础上，研究人员还进一步研究了如何将挖掘出来的规则推荐给程序员，既可以针对程序员当前的编程任务进行推荐，也可以针对程序中的类型转换进行推荐[9]。

4.3　软件的分析、测试与验证

软件质量保障的主要方法包括静态缺陷查找、测试、修复（定位）、监测等。大量工具的出现与应用是这个方向近期的良好进展。

在静态分析方面，近年来由于大量开源软件的出现，给静态缺陷查找方法的研究提供了很多有利的基础。通过分析大量的开源软件，人们可以发现许多共性的缺陷模式，并可以利用历史版本数据做验证，客观地评估所提出的方法。基于这些方法开发的许多静态分析工具近年来在业界得到了非常广泛的应用。例如 Findbugs/Fortify、Coverify、Lint 系列等。

软件测试与验证工具与环境对提高软件产品质量起到重要的作用。在验证方面的一个代表性前沿工作是 SLAM 项目[10]及其所开发的 SDV（静态驱动程序验证），该工具对设备驱动程序验证去看其是否遵循事先定义的 API 使用规则。为了既做到可分析有一定规模的设备驱动程序又做到低的误报率，SDV 工具针对要去验证的 API 使用规则来为程序创建基于谓词的抽象模型，以及用模型检查产生的反例来细化抽象模型。在自动软件测试工具上，动态符号执行近年来取得了许多进展。动态符号执行通过探索遍历代码中可执行路径来达到覆盖不同语句或分支并查错。两个有较大影响力的基于动态符号执行的工具是：针对基于 . NET 的程序（比如 C#程序）的通用测试工具 Pex[11]，以及针对二进制码形式的程序的安全测试工具 SAGE[12]。

随着软件生命周期的不断延长，软件维护技术日趋重要。准确地了解软件特征和软件实现代码间的追踪关系是进行许多维护活动的前提条件。在长期的软件维护过程中，手工维护这种追踪关系是维护人员很大的负担。从 20 世纪 90 年代起，研究人员提出了一系列自动和半自动恢复软件特征和软件实现代码间的追踪关系的技术，为降低软件维护的负担带来了一定的帮助。具体而言，主要包含两种技术路线：①以可视化的方式展示实现代码中的多种依赖关系，让维护人员通过浏览这些依赖关系确定追踪关系；②通过实际执行和信息检索等技术获取软件特征和软件实现代码间的初步联系，并进一步通过程序分析确定追踪关系。在前一技术路线中，研究人员主要关注如何更好地展示相关的依赖关系；在后一种技术路线中，研究人员主要关注如何更高效地获取初步联系以及更准确地从初步联系中精化出最终的追踪关系。

4.4　基于数据的软件工程方法

基于数据的软件工程旨在通过对过往软件开发过程各个阶段数据的分析和抽象，获得并积累软件开发知识与规律，为今后的软件开发提供及时、有效的信息以辅助开发人员进行决策判断。大型软件开发公司往往积累了大量的私有历史数据，这些历史数据对于指导公司决策具有重要意义，这也是产生基于数据的软件工程的重要动力之一。另一方面，开源世界里存在大量的公共历史数据，它们为基于数据的软件工程提供了数据基

础，进一步促进了基于数据的软件工程的发展。

软件开发过程中积累了丰富的数据，包括开发团队信息和开发过程中使用的各种软件库等。典型的软件库包括历史库（如版本管理库、软件故障库）、运行库（如部署日志）、代码库（如 Sourceforge. net、Google Code）。基于数据的软件工程，以软件开发过程的历史数据为基础，通过数据分析建立数据间的关联，挖掘出针对特定软件系统/软件项目或任意软件系统/软件项目等的有用信息，整理成知识或总结成规律，为将来的软件开发过程给出指导。基于数据的软件工程的研究主要集中于数据的抽取和分析：数据抽取旨在使用各种信息抽取技术，从软件开发过程中的各种历史数据中抽取出有意义的信息或知识；数据分析旨在通过建立数据关联，进行数据分析，获得有指导性的结论。基于数据的软件工程早年体现于挖掘软件工程数据或软件储藏库[13]。这个领域的兴起来自于开源代码项目的流行[14]以及实践中来自于软件开发生命周期各个阶段的任务需求[15]。

不可否认，软件工程的研究自提出以来迄今已经有了很大进展。尽管如此，这些进展对于软件危机的缓解效果距离人们的期望值还很大。这其中，我们对软件的认识还未成熟是主要原因之一。更重要的是，软件的运行环境、应用领域和软件自身的形态也都还处于显著变化的阶段。我们当前对软件的认识很难超越软件自身的发展，因此我们研究的软件工程方法也自然面临完全不同于其他学科的巨大挑战。

5 软件工程在中国

我国的软件工程研究始于 1980 年，经过 30 多年的努力，目前已经取得了一些的显著成果，部分技术成果处于国际前列。

5.1 历史成果

南京大学徐家福带领的团队在 1980 年到 1990 年之间开展了软件开发自动化的研究工作[17]。典型的研究成果包括：对需求分析的自动化支持（结构化编程及面向对象的编程），对系统分析与编程的自动化支持（例如，算法设计的自动化，适应性软件的程序合成与设计），对元层程序转换的自动化支持，等等。

1980 年开始，中科院软件所唐稚松领导的研究团队提出了 XYZ/E。这是全球第一个可执行的时序逻辑语言[16]。XYZ/E 是一个得到实际使用的编程语言，可以直接书写其中自动机的状态转换过程。基于该语言，他们还开发了一个软件工程环境。在这个环境中，不同开发阶段的软件制品可以用一个基于 XYZ/E 的统一框架表示。这个环境还能支持编程的不同范型（例如面向对象）以及特定于领域的编程。

1990 年开始中科院软件所董韫美领导的另外一个团队开始研究形式化规约的获取与复用：MLIRF [18]。MLIRF 的目的是探索人机合作的开发方法，利用计算机帮助软件开发人员。软件开发的目标问题起初是一些不精确、不一致的非形式化描述，如何利用已知的规

约知识，通过验证等获得完整、一致的形式化规约，以用做软件设计与实现的基础？其中的关键点包括基于递归功能的内容密集语言，以及语法演绎（Syntax Deduction）的复用。

近 20 多年来，北京大学杨芙清等致力于开发大规模的软件工程环境。其主要目标是为中国的软件企业提供一系列的高效软件生产技术与工具，并形成了系列的“青鸟”方法与产品[19]。“青鸟”工程从最初的核心软件工程开发环境开始，逐步向软件工程过程的各个阶段发展，建立起了一系列的开发方法及相应的支持工具。尤其是，构件化、网络化的软件是后期的主要支持对象，并取得了显著成效。

5.2　最新进展

从 2010 年以来，国内更多的研究成果为国际同行所认可。其显著特征是在国际顶级会议（ICSE，FSE 和 ASE）和刊物（TSE 和 ToSEM）上发表论文有了很大的进展。根据我们的统计，国内学者在上述主要会议（长文）与刊物（发表论文总数为 514 篇）上共发表文章 27 篇，其中北京大学 10 篇，南京大学 4 篇，清华大学 3 篇，中科院软件所 3 篇。这些研究成果涵盖缺陷检测与故障定位、软件演化、软件测试与调试、软件社会学等。

近三年国内对于缺陷检测的研究最为热门，这类研究工作的目标是自动地分析出软件中存在的缺陷，以提高软件质量、降低软件维护的成本。清华大学的郑纬民等提出了一个轻量级的检测多线程程序中的数据竞争问题的工具 RACEZ[21]。类似地，中科院软件所的李明树等则主要研究对于多线程程序的模型检测方法，研究重点是如何对程序进行简化以减小模型检测问题的状态空间，同时还能保证检测结果的正确性[22]。北京大学的梅宏等重点研究的是基于缺陷模式的缺陷检测。缺陷模式指明了缺陷代码片段的特征，缺陷检测工具通过将受检程序和缺陷模式进行匹配以发现程序中的缺陷。他们研究了如何自动地获取这些缺陷模式[23]，并进一步研究了如何对缺陷检测工具报告出来的缺陷列表进行排序，以提高程序员排查缺陷的效率[24]。类似地，北京理工大学的牛振东等提出了一个方法以检测代码中不良编程风格的代码片段，并也重点探讨了对这些代码片段的排序方法[25]。西安交通大学的宋擒豹等提出了一个通用的预测软件缺陷的框架，该框架可以对比不同的缺陷预测方法以帮助程序员找到最适合自己的工具[26]。

此外，在缺陷检测领域中，另一个比较重要的研究课题是缺陷报告系统。软件的使用者向缺陷报告系统中提交缺陷报告，软件开发者根据缺陷报告修复相关的缺陷。微软亚洲研究院的张冬梅等研究了缺陷报告的聚类问题，以减少软件开发者排查缺陷的工作量[27]。大连理工大学的江贺等指出，不同的开发者对于一个软件项目的贡献不同，他们提出了一个基于缺陷报告系统中的社交网络对软件开发者进行优先级排序的方法，并探讨了如何根据程序员的排序，优化缺陷审核、缺陷的严重程度认定以及缺陷预测等问题[28]。

对于已知的软件缺陷，程序员要通过软件调试等方式排查、修复缺陷。调试缺陷的一个重要环节是缺陷的重现。微软亚洲研究院的张峥等提出了一个缺陷自动重现工具 iTarget，该工具可以自动识别出程序和程序执行环境之间的最佳接口，并在该接口处记录程序和环境之间的交互数据以帮助缺陷的重现[29]。此外，调试缺陷的前提在于找到缺

陷所对应的代码片段，也就是缺陷定位问题。清华大学的张宏宇等主要研究了这个问题，他们提出可以通过利用缺陷和代码文件之间的文本相似度信息进行定位[30]，也可以通过挖掘软件版本库和缺陷报告库中的关联信息以推荐出缺陷和程序片段的对应关系[31]。微软亚洲研究院的张冬梅等更进一步对性能调试问题进行了研究，他们提出的方法从 Windows 系统中收集的大量存在性能问题的程序执行轨迹中挖掘有效的信息以帮助程序员进行性能调试[32]。

除了缺陷检测技术以外，软件测试也是发现软件缺陷、保证软件质量的重要手段。北京大学的梅宏等对于测试集压缩技术[33]和变体测试中的变体选择[34]问题进行了研究，他们研究的目标在于如何在保证测试集发现缺陷的能力不变的情况下，尽可能地缩小测试集，以降低测试的开销。类似的，南京大学的聂长海等人研究了在组合测试中，如何使用最小的代价发现最多的错误，并提出了“最小引发错误策略”的理论[35]。南京大学的徐宝文等进一步研究了测试用例对应的布尔范式以及布尔范式的变种之间的缺陷层次结构[36]。

随着软件周期的推移，软件会在不同层次上进行演化，如何在演化过程中避免引入缺陷也是研究人员重点关注的问题。在软件演化过程中，当程序所使用的框架发生演化时，程序也需要做相应的调整。北京大学的梅宏等提出了通过分析软件演化历史以抽取当框架发生演化时的程序自动调整的规则[37]。此外，在软件维护和演化过程中，程序员经常需要将一个改动应用到代码中的多个相似的地方。但是由于代码的复杂性，程序员要找到所有这些相似的地方是很困难的。北京大学的梅宏等提出的方法通过分析软件构件之间的依赖关系，找到所有需要做类似更改的地方，以减少程序员的工作量[38]。

软件的在线演化要求软件在运行过程中进行演化，演化的动作不能够影响软件的运行。复旦大学的臧斌宇等提出了一个名为 POLUS 的软件演化工具，能够迭代地将一个运行中的多线程应用演化到新版本，而且仅产生非常低的开销[39]。南京大学的吕健等则重点研究了分布式系统演化的问题，他们提出了一个分布式算法，以保证分布式系统演化过程中的版本一致性[40]。

代码的合成也是近几年的研究热点之一，代码的自动合成可以减少程序员编程的工作量。上海交大的赵建军等提出了一个方法，该方法能为使用 API 编程的程序员自动的推荐出 API 调用的参数[41]。此外，代码合成在软件迁移过程中也非常有用。一个软件往往需要开发多个语言的版本，为了方便软件在不同语言之间进行迁移，研究人员开发了自动迁移工具。中科院的钟浩等人提出的方法挖掘不同语言的 API 之间的映射规则[42]，基于这些规则，自动迁移工具就可以生成不同语言下的软件版本。

此外，北京大学的梅宏等还对软件的国际化问题进行了研究。他们提出了一个方法可以识别出一个软件应用中需要进行语言翻译的字符串常量[43]，以辅助程序员更方便地进行软件的国际化。

在软件开发过程中，起到主导作用的是程序员。程序员的专业水平、对项目的熟悉程度，甚至程序员的沟通能力都会对软件的开发和维护过程产生重大的影响。北京大学的周明辉等人对于软件开发过程中的程序员的学习曲线、社交活动，以及对软件项目的热衷程度等进行了观察和学习，以发现一些对于软件开发有指导意义的规律[44~46]。南京

大学的荣国平等人则对于软件教育问题进行了研究，他们试图探究对于一个较大的班级如何保证每个学生的足够的参与性的问题的答案。他们提出当以两个学生为单位进行学习和完成作业时达到的学习效果要优于单个同学独自学习的效果，并利用南京大学的暑期学校的一个班级进行了实验和验证[47]。

6　面临的挑战

作为一门学科，软件工程至今仍然处于快速成长时期。我们认为，如下四方面是推动软件向前发展的主要力量：1）更符合人类思维模式的软件模型；2）支持高效高质量的软件开发；3）支持高效能、高可靠、易管理的软件运行；4）更全面、有指导意义的质量评价。不仅如此，软件归根到底是人类智力的产物，不是一种客观存在，而且至今软件形态仍然处于不断演化之中。软件的上述特点使得关于软件的研究面临着许多与传统学科不同的挑战。

6.1　技术上的挑战

关注属性的变化。随着软件在更多的领域得到应用，更多功能之外的软件属性被关注。因此，在软件质量的基础之上，人们开始关注软件的可信性。可信性的提出扩展了原来质量属性的内涵，将人的期望放到一个更高的位置上。这就需要有更多的技术来保障、度量这些新的属性。

运行环境的变化。伴随着不断扩大的应用领域，软件还处于不断变化的环境中，这既包括硬件运行平台（单机、计算机网络、物联网等）的发展，也包括软件运行平台（库、框架、中间件、PaaS 等）的发展。这些变化使得研究成果具有显著的实效性：一些结果在当前环境中是有效的，但如果环境发生了变化，其效果就可能与预期发生偏离。

6.2　研究方法的挑战

软件的研究成果众多且发散。以编程语言为例，有记录的编程语言目前就达 9000 多种。如何评价这些研究成果？评价一个方法的科学方法是重现一下方法的过程，看是否得到声明的结果。但目前的许多研究结果重复困难。以发表在软件工程顶级刊物《软件工程与方法学》（TOSEM）2001 年到 2006 年上的文章为例，60% 的成果体现为具体的工具，但只有 20% 的工具能够安装起来[48]。

对于软件制品的研究相对接近传统学科。其中，以经典算法与数据结构的研究为典型代表。软件的动态测试与静态分析也相对客观一些，前者有大量的测试样例库，后者有大量的开源软件作为基准参考。而对于涉及开发过程的研究，例如，新的需求获取方法，新的建模方法，新的开发模型等，对于研究成果的验证则困难得多。一方面，研究

人员很难获取真实项目中软件开发过程中产生的数据；另一方面，有说服力的评价需要程序员参与验证。由于程序员水平存在差异、理想实验环境的搭建成本高，验证时间又往往比较长，因此目前采用传统验证方法的研究很少见。值得关注的是，心理学科的研究对象是人，在实验方法方面有较系统的成果，非常值得软件工程研究人员借鉴。

6.3 学科基础的挑战

科学是工程的基础，为工程质量、工程进度、工程成本的预测与控制提供基本指导。目前在软件学科基础方面的研究还有很大的空间。自动机与形式化方法曾经在编译、验证等方面发挥了很大的作用，有力地推动了软件的发展，形成了重要的软件学科基础内容。近年来，尽管在需求分析、建模、验证等方面仍有许多形式化的方法被提出，但研究成果缺乏较大的突破。尤其是，近期许多被广泛使用的网络编程语言（PHP、JSP 等）缺乏严格的语言理论基础，给质量保障等带来新的挑战。以 PHP 为例，其本义为“个人网页工具”（Personal Home Page Tools），但目前成为仅次于 C、Java、C ++ 的主流编程语言之一。这是目前面临的现状，但对于研究人员也是一种机会。

软件开发所具有的“艺术”与“工程”双层特征存在着一些冲突。这些冲突有些可以解决，有些可以缓解，有些则长期存在。这就决定了在“软件工程”领域，尽管在不断取得一些显著的进展，但是走的是一条艰辛的探索道路。

7 展望

计算机网络的发展给计算机的各个学科发展都带来了显著的影响。软件工程也不例外。

7.1 全球化的软件开发技术

互联网和通信设备的发展，淡化了地域对软件开发的影响，促进了全球各地域之间联系纽带的建立，进而使得全球化的软件开发成为可能。全球化的软件开发，有效地利用各地区的资源，将软件项目分解并把任务分配给各地区完成，进而提高了软件开发的效率，降低了软件项目的开发成本。某种意义上，全球化的软件开发蕴涵了网络化软件开发和外包开发的双重含义。

全球化的软件开发中的重大挑战在于跨地域的协调。开发团队的地理位置距离越远，团队的协调障碍越大。全球化的软件开发，因其开发团队的地理位置分布，显然会带来不同程度的协调问题。协调问题的具体表现有：有效沟通少、成员状态不透明、开发过程工具等不兼容。全球化的软件开发技术，主要研究全球化软件开发中的通信、协同、协调和知识管理（如群体智慧等），以及支持全球化软件开发的方法、过程和工具等。目前，全球化的软件开发日益成为软件领域（包括学术界和工业界）关注的焦点之一，

也成为很多国际会议的主题。

7.2 基于网络的协同式开发

软件开发是一组人的活动，如何支持相关人员的协作，以提高软件的开发效率值得关注。许多软件公司都在推出类似的产品，例如 Mozilla 的协同式云编程环境 Bespin，IBM 的 Jazz 系统，微软的 Visual Studio Team Foundation Server 等。我国有学者进一步提出了“群体软件工程”的概念：云计算与网联网的发展衍生了超量的信息系统。能不能参考 Facebook 的应用数据处理、APP Store 的应用程序到 TopCoder 的简单服务系统开发方式，使用群体方式来开发一个复杂系统？群体软件工程是一种新的软件工程：面对超量的信息系统，采用群体的研发，群体竞争的研发方式，产生安全可靠软件的一种新的工程。这是它的目标。群体软件工程的开发原则：使用者即设计者，使用者即开发者，使用者即维护者。因为要群体参加，又要保证安全，所以整个的体系结构必须是多层的体系结构。其中的原理包括：屏蔽原理、群件组合服务原理、用户身份的多重性原理、开发者的竞争选择原理，竞争性测试和对抗性安全等（李未，根据 2012 云计算大会讲稿整理）。

7.3 云计算带来的新机遇

云计算出现后，大量的软件将运行在集中的服务器上。如果能继续延伸一下：在开发阶段，程序员也是利用云服务化的各种开发工具进行的，那就给软件生命周期的各种数据获取带来了很大的方便，进而有望逐步建立开发过程的学科基础。

1）获取软件制品。目前已经有许多开源代码库（CVS，SVN 等）、项目跟踪系统（Bugzilla，Issue Tracker System 等），并已经有许多工作（Mining Software Repository）研究分析如下问题：共性的缺陷模式、缺陷是如何引入、消除的？软件是如何演化的？等。

2）获取软件运行状况信息。在云计算模式下，完全可以将用户看做一类特殊的测试人员，用户使用一次，就相当于帮助测试一次。在传统分发方式下，这类测试的结果获取困难。许多人在所用的软件出现错误时，都遇到过类似于“是否将出错信息发给生产商?”这样的信息。这就是生产商在收集特殊的测试数据，以便于找出软件中存在的缺陷。云计算模式中这类结果的获取就方便多了：软件本来就运行在服务的运营商那里。有了这些数据，分析出错根源就方便多了。

3）获取用户的行为。类似地，传统方式获取用户的行为非常困难。微软曾经发去过“用户体验活动”，以获取这类数据。当软件工具部署在云中时，获取这些数据也十分方便，有了这些数据，开发者就可以知道一些重要信息：用户一般如何使用软件？哪些功能最受用户欢迎？这对于后续版本的演化是非常重要的。

4）获取开发人员的行为。目前已经有一些研究分析这类数据（包括 SVN 中的信

息），但检查点过粗，可能一天只有一两条数据。如果能够通过“云编程”的方式，获取细粒度的行为信息，就可以将开发者的主观行为客观地记录下来。这对于了解不同类型用户的开发模式、不同开发方法的对比是非常有价值的。

随着运行平台及使用模式的发展，新形态的软件也在不断出现，导致软件的种类越来越多。对于不同类型的软件，既要努力探索其中共性的方法，又要分别研究各自的特点，寻找适合具体类型软件的方法。总的来看，目前软件工程领域所取得的进展还非常有限，更多、更有价值的内容还需要我们去进一步探索。

供稿：王千祥，张路，谢涛。特别感谢梅宏教授提出的宝贵建议。

参考文献

[1] 梅宏，王千祥，张路，王戟．软件分析技术进展[J]，计算机学报，2009：1697-1710.

[2] 弗雷德里克 布鲁克斯．人月神话[M]．北京：清华大学出版社，2002.

[3] 张效祥，等．计算机百科全书[M]．2 版．北京：清华大学出版社，2005.

[4] 芬顿，等．软件度量[M]．2 版．北京：机械工业出版社，2004.

[5] Beck, Kent (1999). Embracing Change with Extreme Programming. Computer 32 (10): 70-77.

[6] Beck, Kent, et al. . “Manifesto for Agile Software Development”. Agile Alliance. Retrieved 14 June 2010.

[7] Object Management Group, Model Driven Architecture, www. omg. org, 2002.

[8] 蔡维德，白晓颖，陈以农．浅谈深析面向服务的软件工程[M]．清华大学出版社，2008.

[9] Hao Zhong, Lu Zhang, Tao Xie, Hong Mei . Inferring Resource Specifications from Natural Language API Documentation, Proc. of International Conference on Automated Software Engineering (ASE 09), pp. 307-318.

[10] T Ball, V Levin, S K Rajamani. A Decade of Software Model Checking with SLAM, Communications of the ACM, 2011, 54(7): 68-76.

[11] N Tillmann, J de Halleux. Proc. of International Conference on Tests and Proofs (TAP 2008), pages 134-153, 2008.

[12] P Godefroid, M Y Levin, D Molnar. SAGE: Whitebox Fuzzing for Security Testing. ACM Queue, Volume 10 Issue 1, Pages 20, January 2012.

[13] T Xie, S Thummalapenta, D Lo, and C Liu. Data Mining for Software Engineering. IEEE Computer, 42 (8), pp. 35-42, August 2009.

[14] A Mockus, R T Fielding, J Herbsleb. Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3): 1-38, July 2002.

[15] A E Hassan, T Xie. Software Intelligence: Future of Mining Software Engineering Data. In Proceedings of FSE/SDP Workshop on the Future of Software Engineering Research (FoSER 2010), pages 161-166, November 2010.

[16] Tang, C-S. Toward a Unified Logic Basis for Programming Languages. IFIP Congress 1983, pp. 425-429.

[17] Xu J, Lu J. Software Languages and Their Implementation, Scientific Publishing House, 2000. (in Chinese)

[18] Dong, Y, K Li, H Chen, et al., Design and implementation of the formal specification acquisition system SAQ, Conf. Software: Theory and Practice, IFIP 16th World Computer Congress 2000. Beijing, 2000, 201-211.

[19] Yang, F, Mei, H. Research on Technology for Industrialization Production of Software—Practice of JB (Jade Bird) Project, Symposium of Sino-American Engineering Technology, Oct. 1997, Beijing, 190-200.

[20] Zannier, Melnik. Maurer evaluates ICSE empirical studies, ICSE'06.

[21] T Sheng, N Vachharajani, S Eranian, R Hundt, W Chen, W Zheng. RACEZ: A Lightweight and Non-Invasive Race Detection Tool for Production Applications. Proc. of International Conference on Software Engineering (ICSE 11), pages 401-410, 2011.

[22] Qiusong Yang, Mingshu Li. A Cut-off Approach for Bounded Verification of Parameterized Systems. Proc. of International Conference on Software Engineering (ICSE 10), pages 345-354, 2010.

[23] Q Wu, G Liang, Q Wang, T Xie, and H Mei. Iterative Mining of Resource-Releasing Specifications. Proc. of International Conference on Automated Software Engineering (ASE 11), pages 233-242, 2011.

[24] G Liang, L Wu, Q Wu, Q Wang, T Xie, H Mei. Automatic Construction of An Effective Training Set for Prioritizing Static Analysis Warnings. Proc. of International Conference on Automated Software Engineering (ASE 10), pages 93-102, 2010.

[25] H Liu, Z Ma, W Shao, Z Niu. Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort. IEEE Transactions on Software Engineering (TSE 12), 2012, 38(1): 220-235.

[26] Q Song , Z Jia, M Shepperd , S Ying, J Liu. A General Software Defect-Proneness Prediction Framework. IEEE Transactions on Software Engineering (TSE 11), 2011, 37(3): 356-370, 2011.

[27] Y Dang, R Wu, H Zhang, D Zhang, P Nobel. ReBucket: A Method for Clustering Duplicate Crash Reports Based on Call Stack Similarity. Proc. of International Conference on Software Engineering (ICSE 12), pages 1084-1093, 2012.

[28] J Xuan, H Jiang, Z Ren, W Zou. Developer Prioritization in Bug Repositories. Proc. of International Conference on Software Engineering (ICSE 12), pages 25-35, 2012.

[29] M Wu, F Long, X Wang, Z Xu, H Lin, X Liu, Z Guo, H Guo, L Zhou, Z Zhang. Language-based Replay via Data Flow Cut. Proc. of International Symposium on Foundations of Software Engineering (FSE 10), pages 197-206, 2010.

[30] J Zhou, H Zhang, D Lo. Where should the bugs be fixed? Proc. of International Conference on Software Engineering (ICSE 12), pages 14-24, 2012.

[31] R Wu, H Zhang, S Kim, S Cheung. ReLink: recovering links between bugs and changes. Proc. of International Symposium on Foundations of Software Engineering (FSE 11), pages 15-25, 2011.

[32] S Han, Y Dang, S Ge, D Zhang, T Xie. Performance debugging in the large via mining millions of stack traces. Proc. of International Conference on Software Engineering (ICSE 12), pages 145-155, 2012.

[33] D Hao, L Zhang, X Wu, H Mei, G Rothermel. On-demand test suite reduction. Proc. of International Conference on Software Engineering (ICSE 12), pages 738-748, 2012.

[34] L Zhang, S Hou, T Xie, H Mei. Is operator-based mutant selection superior to random mutant selection? Proc. of International Conference on Software Engineering (ICSE 10), pages 435-444, 2010.

[35] C Nie, H Leung. The Minimal Failure-Causing Schema of Combinatorial Testing. ACM Transactions on Software Engineering and Methodology (TOSEM 11), 2011, 20(4).

[36] Z Chen, T Y Chen, B Xu. A revisit of fault class hierarchies in general boolean specifications. ACM Trans-

actions on Software Engineering and Methodology (TOSEM 11), 2011, 20(3).
[37] S Meng, X Wang, L Zhang, H Mei. A history-based matching approach to identification of framework evolution. Proc. of International Conference on Software Engineering (ICSE 12), pages 353-363, 2012.
[38] X Wang, D Lo, J Cheng, L Zhang, H Mei, J X Yu. Matching dependence-related queries in the system dependence graph. Proc. of International Conference on Automated Software Engineering (ASE 10), pages 457-466, 2010.
[39] H Chen, J Yu, C Hang, B Zang, P Yew. Dynamic Software Updating Using a Relaxed Consistency Model. IEEE Transactions on Software Engineering (TSE 11), 2011, 37(5): 679-694.
[40] X Ma, L Baresi, C Ghezzi, V P L Manna, J Lu Version-consistent dynamic reconfiguration of component-based distributed systems. Proc. of International Symposium on Foundations of Software Engineering (FSE 11), pages 245-255, 2011.
[41] C Zhang, J Yang, Y Zhang, J Fan, X Zhang, J Zhao, P Ou. Automatic parameter recommendation for practical API usage. Proc. of International Conference on Software Engineering (ICSE 12), pages 826-836, 2012.
[42] H Zhong, S Thummalapenta, T Xie, L Zhang, Q Wang. Mining API mapping for language migration. Proc. of International Conference on Software Engineering (ICSE 10), pages 195-204, 2010.
[43] X Wang, L Zhang, T Xie, H Mei, Ji Sun. Locating need-to-translate constant strings in web applications. Proc. of International Symposium on Foundations of Software Engineering (FSE 10), pages 87-96, 2010.
[44] M Zhou, A Mockus. Does the Initial Environment Impact the Future of Developers? Proc. of International Conference on Software Engineering (ICSE 11), pages 271-280, 2011.
[45] M Zhou, A Mockus. What make long term contributors: willingness and opportunity in OSS community. Proc. of International Conference on Software Engineering (ICSE 12), pages 518-528, 2012.
[46] M Zhou, A Mockus. Developer fluency: achieving true mastery in software projects. Proc. of International Symposium on Foundations of Software Engineering (FSE 10), pages 137-146, 2010.
[47] G Rong, H Zhang, M Xie, D Shao. Improving PSP education by pairing: an empirical study. Proc. of International Conference on Software Engineering (ICSE 12), pages 1245-1254, 2012.
[48] Carlo Ghezzi, Reflections on 40 + years of software engineering research and beyond an insider's view, ICSE 2009 Keynote, 2009.
[49] D Zhang, T Xie. Software Analytics in Practice: Mini Tutorial. In Proceedings of the 34th International Conference on Software Engineering (ICSE 2012), Software Engineering in Practice, Mini Tutorial, June 2012.

作者简介

王千祥 博士，北京大学信息科学与技术学院教授。CCF 软件工程专业委员会秘书长。主要研究方向包括服务化软件的监测与分析、软件中间件等。

张　路　博士，北京大学信息科学与技术学院教授。CCF 高级会员，2010 年 CCF 青年科学家奖获得者。主要研究方向包括软件分析与测试、软件维护与演化、软件服务工程等。

谢　涛　博士，北京大学信息科学与技术学院访问教授（2011—2012 年）。主要研究方向包括软件分析与测试、基于数据的软件工程等。