软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(1):3−21 [doi: 10.13328/j.cnki.jos.005650]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

软件开发方法发展回顾与展望*

马晓星[1,2], 刘譞哲[3,4], 谢 冰[3,4], 余 萍[1,2], 张 天[1,2], 卜 磊[1,2], 李宣东[1,2]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[2](南京大学 计算机科学与技术系,江苏 南京 210023)

[3](北京大学 信息科学技术学院 软件研究所,北京 100871)

[4](高可信软件技术教育部重点实验室(北京大学),北京 100871)

通讯作者: 李宣东, E-mail: lxd@nju.edu.cn

摘 要: 软件是信息化社会的基础设施,而构造并运用软件的能力成为一种核心竞争力.软件开发方法凝结了系统化的软件构造过程和技术.简要回顾了50年来软件开发方法发展历程中具有重要影响的里程碑,包括基于结构化程序设计和模块化开发的基本方法、面向对象方法、软件复用与构件化方法、面向方面的方法、模型驱动的方法,以及服务化的方法.而后针对 Internet 的发展普及以及人机物融合应用对软件开发方法提出的挑战,介绍了网构软件的研究和探索,并展望未来人机物融合的软件方法和技术.

关键词: 软件开发方法;回顾;展望

中图法分类号: TP311

中文引用格式: 马晓星,刘譞哲,谢冰,余萍,张天,卜磊,李宣东.软件开发方法发展回顾与展望.软件学报,2019,30(1):3−21. http://www.jos.org.cn/1000-9825/5650.htm

英文引用格式: Ma XX, Liu XZ, Xie B, Yu P, Zhang T, Bu L, Li XD. Software development methods: Review and outlook. Ruan Jian Xue Bao/Journal of Software, 2019,30(1):3−21 (in Chinese). http://www.jos.org.cn/1000-9825/5650.htm

Software Development Methods: Review and Outlook

MA Xiao-Xing[1,2], LIU Xuan-Zhe[3,4], XIE Bing[3,4], YU Ping[1,2], ZHANG Tian[1,2], BU Lei[1,2], LI Xuan-Dong[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

[3](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[4](Key Labortory of High Confidence Software Technologies of Ministry of Education (Peking University), Beijing 100871, China)

Abstract: Software constitutes the infrastructure of an information society, and the production and utilization of software systems become a core competency. A software development method embodies a systematic set of processes and techniques enabling the engineering of software systems. This paper briefly reviews some most important software development methods thrived in the last five decades, including fundamental methods based on structured programming and modular development, object-oriented methods, software reuse and component-based methods, aspect-oriented methods, model-driven methods, and service-oriented methods. After that it gives an outlook with an introduction to the research efforts on Internetware and a call for future software development methods for emerging social-cyber-physical systems.

Key words: software development method; review; outlook

---

* 基金项目: 国家自然科学基金委-中国科学院学科发展战略研究项目:软件学科发展战略研究(XK2017XXC01)

Foundation item: NSFC-CAS Joint Research Funds for Scientific Discipline Development Strategies, Software Science and Engineering (XK2017XXC01)

本文由“软件学科发展回顾特刊”特约编辑梅宏教授、金芝教授、郝丹副教授推荐.

收稿时间: 2018-07-31; 修改时间: 2018-08-09; 采用时间: 2018-09-29; jos 在线出版时间: 2018-11-22

CNKI 网络优先出版: 2018-11-23 07:18:01, http://kns.cnki.net/kcms/detail/11.2560.TP.20181123.0717.004.html

随着计算机技术的飞速发展,软件的使能空间得到了广泛和持续的拓展,软件系统的规模和复杂性也随之不断增大,人类正在进入“软件无处不在、软件定义一切、软件使能一切”的时代.开发和演化软件系统,成为人类创造财富、延续文明的重要需求和途径.

软件是人类制造的最复杂的一类制品,是人类大脑思维活动的体现.开发和演化软件系统需要方法,软件开发方法一般指“软件开发过程需要遵循的办法和步骤”[1](软件开发方法在英文中常使用两个不同的术语“Software Development Method”和“Software Development Methodology”,后者亦常翻译为软件开发方法学.后者传统上更强调某种特定的覆盖整个软件开发周期的完整过程和技术体系,近年甚至常被误用为软件开发过程“Software Development Process”的同义词.本文在较宽泛的意义下讨论软件开发方法,并使用前一英译).长期以来,软件开发方法的发展和演变归纳起来主要与 3 个方面相关联.

- 其一是外在因素,包括运行环境(硬件、网络、外设的发展和普及)的驱动、应用需求的牵引、信息领域技术浪潮(例如近年来涌现的大数据、云计算、物联网、虚拟现实、人工智能等等)的推动;
- 其二是内在动力,包括效率、质量和成本.解决软件系统规模和复杂性所导致的问题,从而高效、高质量、低成本地开发和演化软件系统,一直是学术界和工业界追求的共同目标;
- 其三是人本属性.人是开发和演化软件系统的主体,软件开发和演化方法一方面要遵循人的认识规律,另一方面要能够充分激发、调动和管理人力资源.

自 20 世纪 60 年代末提出软件工程概念至今已有 50 年,其间,软件开发方法取得了长足的发展,本文回顾 50 多年来软件开发方法发展过程中的重要里程碑,包括基本方法(结构化程序设计、模块化方法)、面向对象方法、构件化方法、面向方面方法、模型驱动的方法、服务化方法等,进而对“软件无处不在、软件定义一切、软件使能一切”时代下软件开发方法的发展趋势进行展望.

1 软件及其开发方法

究其本质,软件乃是以计算为工具实现应用目标的解决方案.不同于一般物品,软件是一种人工制品[2],同时也是一种纯粹的逻辑制品[3].作为一种人工制品,其需以适应其所处环境的方式完成应用目标;作为逻辑制品,其困难不在于物理限制而在于逻辑构造.因此,软件开发活动本质上不同于传统工程制造.

- 后者在于“造物”;前者可谓“拟人”——即表达人脑思维形成的问题解决方案;
- 后者可有规模效应;而对前者而言,每一个软件系统都是独一无二的创造.

软件开发方法所讨论的是如何高效、低成本地构造高质量的软件,这也是软件工程学科的基本科学问题.Wirth 教授在回顾软件工程发展历史时写到:“如果我们能从过去学到什么的话,那就是计算机科学实际上是一门方法论学科”[4].Brooks 在其“没有银弹”的经典演讲[5]中指出:究其本质而言,软件开发是一项困难的任务,而困难可区分为实质性的(essential)和附属性的(accidental).可以认为,前者来自于软件所要解决问题本身所固有的复杂性和多变性,而后者源自解决问题时所用技术手段和过程步骤方面的不妥.软件开发方法旨在消除附属性困难,并帮助开发者理解和驾驭问题本身的实质性困难.

软件开发方法给出构造软件所需的系统化的过程步骤和技术手段.这种过程和技术的背后是用于指导软件开发这种创造性活动的思维模式.如前所述,软件是人脑力的替代,若把软件比拟为人,则开发方法的思维模式可比拟为软件系统的“世界观”和“方法论”.

- 所谓世界观是指软件如何抽象其所处环境和应用目标;
- 所谓方法论是指软件本身的范型抽象,包括:结构模型,即软件的组成元素、组合方式等静态构成形式;运行机理,即软件各组成元素动态运行及其间交互的机制和原理;构造方式,即如何通过层次化的问题分解和步骤分解来使系统构造成为高效可行的任务;质量保障,即如何定义并改善所构造软件满足目标的程度.

纵观 50 年来软件开发方法研究和实践的发展状况,可以说对软件开发和使用中各方面复杂性的应对是其一个不变的主题.早期的结构化程序设计和模块化方法主要针对的是软件程序本身的复杂性,而后的面向对象

方法在其基础上增强了应对应用领域问题复杂性的能力.进而,面向方面的方法又增强了面向对象方法在处理安全、日志等贯穿性的非功能属性方面的能力.此外,软件复用与构件化方法通过重复使用已有的软件部件来消除重复开发的复杂性;模型驱动的方法则力图通过提高解决问题的抽象层次来应对软件系统日益增长的复杂性;服务化的方法通过中立的访问协议和显式的服务契约解耦服务的提供者和使用者,应对开放异构的互联网环境所带来的复杂性,并简化了软件的获取和使用方式.这些方法的基本思想和实现技术成为当前软件开发实践的主流依托,也是本文回顾的重点.需要说明的是:这些方法之间并非是相互竞争排斥的关系,其历史进程上也未必是线性发展的.本文回顾的顺序组织只是为了便于理解的需要.除以上提及的软件开发方法外,还有一系列的方法从人本属性角度、在软件组织与管理的层面被提出来并产生重要影响,比如原型方法、敏捷方法、运维一体化方法(DevOps)等,由于在本期特刊中另有论文专门讨论,本文不再赘述.

当前,互联网的不断拓展和深化也促成了一种人机物三元融合的发展态势,软件日益成为信息化社会的基础设施,其复杂程度持续提高.本文亦将在此背景下展望软件开发方法进一步的发展趋势.

2 基本方法

20 世纪 50 年代末诞生的高级程序设计语言在很大程度上提高了软件的开发效率,从而进一步催生了对软件系统的需求,软件系统的规模和复杂性也随之增大,各类软件开发方法开始出现并不断发展.软件开发方法是指软件开发过程需要遵循的办法和步骤,早期人们一般性地认为软件开发过程主要包括问题定义、需求分析、设计、编码、测试、维护等阶段,而软件开发方法起步于解决软件设计阶段的问题.软件设计又分为概要设计和详细设计两个不同抽象阶段(分别建立软件结构和程序过程),与之对应的基本方法分别是模块化设计方法和结构化程序设计.

2.1 结构化程序设计

结构化程序设计关注软件详细设计阶段的程序过程(代码)描述.1968 年,Dijkstra 提出了程序中无条件转移语句(goto)有害的观点[6],从而引起了大范围的学术讨论.经过讨论,人们得到共识:goto 语句使得程序的静态结构与程序的动态执行不一致,从而使得程序难以理解和调试.在此基础上,进一步形成了结构化程序设计的主要思想.

结构化程序设计的主要思想是:使用(仅使用)顺序、选择和重复这 3 种结构表示程序过程.由于这 3 种结构具有单入口和单出口特性,因而能够降低程序的复杂性,易于程序理解和维护,提高了可靠性.

2.2 模块化开发方法

模块化方法关注软件概要设计阶段的软件体系结构表示.面对大规模的复杂问题,人们通常的解决办法是分而治之,把大问题看成是若干小问题的集合、把复杂问题看成是若干简单问题的集合.软件开发也是如此,与之对应的是模块化方法.

软件概要设计就是要把软件系统的需求(功能)对应到软件系统的各个组成部分,这些组成部分称为模块.模块化是指把软件系统划分成独立命名和可独立访问的模块,每个模块完成一个子功能,它们集成到一起满足软件的整体功能需求.实现模块化的手段是抽象和信息隐蔽:抽象是指抽出事物的本质特性而暂时不考虑它们的细节;信息隐蔽意指应该这样设计和确定模块,使得一个模块内包含的信息(过程和数据)对于不需要这些信息的模块来说是不可访问的.模块化方法强调模块独立性,模块独立是指开发具有独立功能并且与其他模块之间没有过多相互作用的模块;模块独立的意义在于功能分割、简化接口、易于测试和维护、易于多人合作开发同一系统.

3 面向对象方法

复杂性是软件开发过程中所固有的特质[7],而软件开发方法的核心在于帮助开发者驾驭这种复杂性.结构化程序设计和早期的模块化方法围绕功能构造系统,其本质是功能分解、过程抽象、自顶向下、逐步求精.这

种方法有助于帮助开发者驾驭软件设计和实现的复杂性,但在应对应用需求的复杂性方面考虑不多.随着软件应用范围的拓展和复杂程度的提高,以及软件需求变化导致的软件演化日益频繁,面向对象的软件开发方法逐渐成为主流.

3.1 基本思想

面向对象的软件系统通过一组对象的交互来完成系统的功能.对象是数据及其所允许操作的封装体,是应用领域现实实体的软件抽象.面向对象的软件构造乃是基于系统所操作之对象类型,而非系统需实现之功能,来架构系统的途径[8].面向对象方法的实施步骤包含了面向对象分析、面向对象设计和面向对象实现等.

- 面向对象分析主要包括识别应用领域应被抽象的对象,并分类组织这些对象,形成对象概念模型;刻画这些对象的内部行为和外部交互,以及系统功能目标何以因之达成,形成用例模型;
- 面向对象的设计要在系统各种实施条件的约束下,给出上述概念模型和用例模型的实现方案,包括确定对象模型的操作、设计实现操作的算法、优化数据的访问、调整类结构以提高继承性等,形成面向实现的类模型、对象交互模型、对象状态模型和存储模型等;
- 面向对象实现通常使用面向对象程序设计语言和对象持久化设施等将上述设计具体地加以实现.

在应对需求复杂性方面,面向对象的软件开发方法通过建立与现实世界中的实体、概念、关系和结构直接对应的软件抽象来刻画需求,并支持该软件抽象在需求、设计、实现之间的无缝过渡,有助于弥合问题空间与解空间之间的语义鸿沟.在应对需求变化方面,结构化方法下功能的变化将导致如此设计的系统结构发生较大的变化.而应用领域的概念和结构远比应用功能更稳定.因此,较之结构化方法,面向对象方法开发的软件具有更好的结构稳定性、可修改性和可复用性.

3.2 核心机制与设计原则

面向对象方法的核心机制包括封装、继承和多态.

- 数据抽象提供了面向对象计算的起点,即定义数据类型和施加于该类型对象上的操作,并限定了对象的值只能通过使用这些操作才能修改和观察,从而自然地形成模块封装和信息隐藏.面向对象通过类比发现对象间的相似性,即对象间的共同属性,是构成对象类的依据;
- 继承是类与类之间的一种层次关系,是实现利用可复用软件构件构成系统的有效语言机制.相比过程复用,面向对象是对数据及其上操作的整体复用,而非仅复用操作功能.对象间的相互联系是通过传递“消息”来完成的.通过对象之间的消息通信驱动对象执行一系列的操作,从而完成某一任务;
- 多态性是指相同的操作或函数、过程可作用于多种类型的对象上并获得不同的结果.不同的对象收到同一消息可以产生不同的结果.多态性增强了软件的灵活性和可复用性.综上,面向对象的设计可以进一步总结为把软件系统构作成类(或谓抽象数据类型实现)的结构化集合[9].

为了使开发的软件具有能够快速适应变化的敏捷性,必须遵循面向对象的设计原则[10].其中,

- 开放封闭原则是面向对象设计的核心,即:对扩展开放,对修改封闭[8].Liskov 提出的替换法则指出,子类应该可以在任何地方替换它的父类[11];
- 依赖倒置原则要求面向对象的设计要对结构化的方法进行倒置,即高层模块不应依赖低层模块,两者都应依赖于抽象.依赖关系的倒置正是好的面向对象设计的标志所在;
- 单一职责原则要求类的职责单一,引起类变化的原因单一,这是实现软件灵活性的前提;
- 接口隔离原则强调接口的高内聚和低耦合;
- 迪米特原则要求控制对象之间的信息流量、流向及信息的影响,即注意信息的隐藏;
- 组合/聚合复用原则指出:相比于使用继承来实现复用,更应优先使用组合/聚合的复用方式.通过对象的组合/聚合,可以在运行时动态地配置新对象的功能,并防止类层次规模的爆炸性增长.

3.3 源流与发展

尽管 MIT 的一些学者早在 20 世纪 60 年代初就开始使用对象这一概念,公认的面向对象的思想肇始于 Dahl

和 Nygaard 设计的 Simula 67 语言[12],该语言已完整提出了对象、类、继承、多态等面向对象程序设计的核心概念.并不令人意外,Simula 语言原本用于计算机模拟离散事件系统,其基本理念被广泛用于通用的程序设计和软件开发仍有赖于此后二三十年间的软件思想、方法和技术进展的推动.20 世纪 70 年代,Parnas 提出了模块化与信息隐蔽的基本思想[13];Hoare 和 Liskov 等人引领了数据抽象和抽象数据类型的研究[14,15];Chen 发表了著名的实体关系模型的论文[16],推动了对应用领域现实世界的概念建模;Kay 领导开发的 Smalltalk-80[17]推动了面向对象程序设计语言的发展和广泛应用.

随着面向对象程序设计的日益普及,20 世纪 80 年代末和 90 年代初涌现了一批面向对象的软件分析和设计方法,其中包括:Booch 于 1990 年提出的 Booch 方法[7];1989 年,Coad 和 Yourdon 提出的 Coad 方法[18,19];1991 年,由 Rumbaugh 等 5 人提出的 OMT 方法[20];Jacobson 于 1992 年提出的 OOSE 方法[21]等.此后,统一建模语言(UML)不仅统一了 Booch、Rumbaugh 和 Jacobson 的表示方法,而且对其作了进一步的发展,并最终统一为大众所接受的标准建模语言.UML 不仅支持面向对象的分析与设计,还支持从业务建模、需求获取、分析、设计、实现、测试到部署的不断迭代软件开发活动,在此基础上,形成了所谓统一的软件开发过程[22].

在面向对象设计技术发展方面,对微观对象式设计经验的总结和升华形成了设计模式,而对宏观系统架构的关注催生了对软件体系结构研究.1994 年,Gamma、Helm、Johnson 和 Vlissides 的著作《Design Patterns: Elements of Reusable Object-Oriented Software》共总结了三大类型(创建型、结构型和行为型)共计 23 种设计模式[23].设计模式代表了面向对象设计和实现的最佳实践,正确应用设计模式可以使得系统实现更易复用、更易理解、更易维护,并且更加可靠.软件体系结构从比类更大的粒度、更高的抽象层次上刻画系统的构件组成、构件之间的关联结构及其交互行为,并给出这种宏观设计的动机,即其在功能、非功能目标以及各种制约因素之间的权衡取舍[24].

在面向对象需求分析方面,随着软件应用的日益深入,软件本身亦应被考虑为现实世界的重要组成部分.因而,仅通过模拟现实世界实体的结构和行为不足以指导软件的构造.为此,必须考虑包括软件本身在内的整个系统如何共同满足组织机构的应用目标.这些应用目标不限于通常的面向对象分析所擅长的功能性需求,也包含用户体验、经济性、安全性、适应性等非功能需求.在这方面,目标导向的需求工程给出了面向对象分析的补充[25].目标导向的需求工程考虑应用目标的层次分解,分析不同目标选择,管理目标之间协同和冲突关系,最终将应用目标落实到面向对象分析可应对的具体需求上.

4 面向方面的方法

随着软件系统规模的扩大,其复杂性也急速增长且难以控制.以日志、安全等为典型代表的非功能性属性在常规的面向对象开发当中分布于业务逻辑的各个角落,难以有效统一处理,给软件的理解与维护带来较大障碍.面向方面方法作为面向对象方法的一个有效补充,将相关属性统一为横切关注点进行理解与整理,并将其抽象为“方面”这一概念,从而可以一体化设计与实现,大幅度增加了代码的可理解性与可维护能力.

4.1 面向方面程序设计

随着软件系统规模的快速扩大,软件开发过程中,程序开发人员频繁遇到特定属性散布于多个功能模块,导致难以统一维护、更新与管理的情况.以系统级关注点为例,如日志记录、安全检测等,它们与系统基本任务属性之间是横切的关系,分布在各个功能模块之中.

传统的面向对象程序设计(object-oriented programming,简称 OOP)方法主要着眼于系统核心业务需求,将数据和对应操作紧密地连结并封闭在一起.因此,使用面向对象方法实现上文所述的日志记录、安全检测等任务,自然就会将其与相关对象封装在各自的类中,从而导致处理这些关注点的代码在程序中多次重复出现.可以想象,这些关注点的代码实现其实是高度类似甚或一致的.同样的代码在整个系统中散落且频繁出现,会很容易降低程序的可理解性,而在面对后期潜在的需求变更时,其维护性也会受到很大影响,进而影响系统的质量.这也就是所谓的代码分散(code scattering)现象[26].相对应地,另一个广为人知的问题则是代码缠结(code tangling)[26],即同一个模块的代码中要同时处理安全、功能、性能等各类关注点.可想而知,这样的代码,其可读

性和可维护性也自然会存在较大问题.

上述问题由来以久,Dijkstra 就曾在《A Discipline of Programming》一书中提到“关注点分离(separation of concern)”的概念,即每次处理某一个特定的关注点相关部分.而随着系统规模的快速扩大,代码分散与代码缠结等问题使得“关注点分离”更加必要且迫切.针对这一问题,Kiczales 等人在 1997 年欧洲面向对象编程大会(ECOOP'97)上提出了面向方面的程序设计(aspect oriented programming,简称 AOP)[27,28]这一思想.相比于 OOP,AOP 把系统关注点分为核心关注点与横切关注点(crosscutting concern)两类:核心关注点即业务处理中主要商业逻辑与流程;而横切关注点则是分布在各核心关注点内的共享关注点,如上文所提到的日志、安全等.

AOP 将上文中这些影响了多个类的横切关注点封装到一个可重用模块,命名为“Aspect”,即方面.在此基础上,进一步提供了类和方面的自动编织(weave)机制,将横切关注点对应的方面编织到特点的连接点(join point)位置,实现代码整合.通过对这些“方面”的封装以及方面与类的自动编织来减少系统的重复代码,降低模块间的耦合度,并有利于未来的可操作性和可维护性.

显然,AOP 是在 OOP 的基础上针对“关注点分离”这一目标的一个有力补充.在 OOP 处理核心关注点及其业务逻辑的基础上,AOP 以一种模块化的方式来理解、封装与处理横切关注点相关“方面”.然后,在运行或编译时应用这些方面,从而实现与原核心业务逻辑的结合.这一机制使横切关注点的实现更加模块化,降低了代码的理解和维护难度,并提高了系统适应不断变化的需求的能力.

4.2 面向方面方法

长期以来,AOP 思想一直受到了学术界和工业界的共同关注,其思想已被引入了需求分析、代码实现、测试维护等各个阶段,并衍生出面向方面软件开发(aspect-oriented software development,简称 AOSD)、面向方面需求工程(aspect-oriented requirement engineering,简称 AORE)等多个方向及子研究领域.同时也出现了以 AspectJ[28]、AspectC[29]及 AspectC++[30]等为代表的面向方面程序设计语言.其中,AspectJ 是目前使用得最为广泛的 AOP 语言,其是对 Java 程序设计语言面向方面的扩展,经编译后生成的类文件可以直接在 Java 虚拟机上执行,因此,其也得到了相关领域的广泛应用与实践.在软件开发的各个阶段,如需求分析、建模、测试与验证、重构与应用等方面,AOP 的相关思想和技术也得到不断的发展.

- 面向方面需求分析

需求分析是软件开发的首要步骤,只有在得到正确需求的情况下才可以进行后续开发.AOP 的主要出发点即横切关注点提取,因此,如何提取与表达横切关注点是 AOP 相关领域的重点关注问题.在横切关注点提取方面,如何对需求文档进行分析并自动提取是一个热点方向.EA-Mine[31]是一个采用自然语言技术寻找横切点的代表性工具.类似地,文献[32]采用词汇链分析技术来识别文本需求中的 Aspect.而在表达横切关注点方面,则存在基于状态机的表达[33,34]、基于用例图扩充[35]等多种方法来具体描述相关横切需求.这部分内容将在下面的面向方面建模中进一步展开讨论.

- 面向方面建模

以 UML 为代表的建模工具主要关注的是系统的核心业务逻辑,对于横切关注点缺乏必要的支撑.因此,如何对 UML 进行扩展[36]以支持横切关注点建模,是相关领域的研究热点所在.相关扩展可分为轻量级、重量级两类:所谓轻量级即利用 UML 所提供的 Stereotype、Tagged Value、Constraint 等扩展机制进行模型扩展,如文献[37,38]等,其优点在于可直接利用 UML 建模工具进行建模;而重量级扩展则主要在元模型层面进行直接扩展,给出 Aspect 建模的新型建模元素[39].这种方法所建立的模型更加直观,但却需要对应的工具支撑.相关方向存在大量工作分布在状态机图、用例图、顺序图、交互概观图乃至活动图以及 Petri 网等多种建模语言上.受篇幅所限,此处不再赘述.感兴趣的读者可以阅读 Wimmer 等人的综述[40],对其发展进行详细的了解.

- 面向方面测试与验证

系统正确性测试与验证是保障系统质量的重要手段.面向方向方法也给相关问题带来了新的挑战,如横切关注点实现的正确性、一致性等.当然,直接把经过编织后的代码当作常规代码进行测试[41]是一种可行手段.除此之外,也仍有大量工作从 AOP 的问题特性出发去研制对应的测试方法,包括面向方面的单元测试[42]、集成测

试[43]、回归测试[44,45]等,并研发了相关工具,如 Wrasp[41]、Raspect[46]、EAT[47]、Rambutans[48]等.在验证方向上,也有系列工作采用模型检验技术验证集成后模型与需求的一致性[49],以及验证编织后系统中是否含有死锁等不希望行为等[50].

- 面向方面的重构与应用

除了上述几个方向之外,将面向方面的思想利用到系统重构领域,特别是针对 Aspect 来基于横切关注点进行重构,是相关领域的热点所在[51].其中的主要问题自然就关注在两个层面:如何发现与识别横切关注点、如何基于相关横切点进行重构等[52−54].

在上述系列开发工作的基础上,AOP 思想也在各类领域得到了大量的实际应用,如安全[55,56]、数据存储[57,58]、云计算[59]、竞争检测[60]等.国内相关研究者也在操作系统[61]、构件化开发[62]、轨道计算[63]等领域上展开了面向方面思想的应用与尝试.受篇幅所限,以上工作仅为相关应用的一小部分,近几年仍然可以持续看到 AOP 思想在各类领域的新型应用持续出现.感兴趣的读者可对相关问题进一步加以关注.

5 软件复用与构件化方法

复用是提高效率的基本途径,软件复用是指重复使用已有软件的构件.长期以来,复用一直是软件技术和产业发展的重要关注点,软件复用不仅能够提高开发效率,而且由于使用的越多就越容易发现错误,所以能够保障质量.一般而言,重复使用最为核心的构件是软件代码.程序语言大都提供了使用外部构件的能力,汇编语言的子程序(subroutine)、结构化程序设计中的函数模块(function)以及面向对象程序中的类(class),都是可以复用的基本构件.1968 年,Mcllroy 在 NATO 软件工程会议上发表的论文“大量生产的软件构件”提出了大规模复用的考虑[64].早期具有代表性的工作是美国自然科学基金会组织构建的数学函数库,在 20 世纪 80 年代中期,基本上完成了所有可能的数学函数的子程序,使得在程序中应用各种类数学函数都可以直接复用.从 20 世纪 90 年代中期开始,伴随着面向对象技术开发技术的成熟,类(class)因其直接对应客观世界的实体特性,更易于成为复用的组织单元,而类间结构和联系也体现出了客观实体的关联关系.可以说,面向对象分析设计方法的成熟,使得软件复用可以在更全面的范围内得以实施.

软件复用被视为解决软件危机、提高软件生产效率和软件质量的现实可行的途径[65,66].一般而言,软件开发中都会考虑复用以前的工作基础,例如复用部分程序代码、使用相同领域有过开发经验的人员等等.系统化的软件复用则特指将软件复用全过程、全技术紧密结合的开发过程,使软件复用从早期的关注代码复用逐步发展到基于复用的软件开发全过程,可以包括 3 个主要阶段.

1) 有目的地开发为复用而做的软件(development for reuse);
2) 管理已有的可复用软件(management for reuse);
3) 复用已有的软件开发新软件(development by reuse).

这时的软件复用体现了其核心思想,即:软件系统的开发以现有的工作为基础,充分利用以往软件系统开发过程中积累的知识和经验进行新的软件系统的开发.

5.1 基于构件的软件开发和复用

基于构件的软件开发(component based software development)便是一种典型的软件复用形式.基于构件的软件开发将软件的生产模式从传统的软件编码工作转换为以软件构件为基础的系统集成和组装.软件构件充当基本复用对象的角色,软件构件技术是软件复用技术的核心和基础.构件是指软件系统中具有相对独立功能、可以明确辨识、接口由契约指定、与语境有明显依赖关系、可独立部署且多由第三方提供的可组装软件实体[67].构件模型是构件本质特征及构件间关系的抽象描述.构件模型中,实际包括软件体系结构(software architecture)和构件(component)两部分的定义.

在基于构件的软件复用中,着重研究了 3 方面的技术.

- 一是如何识别可以复用的成分:针对特定领域来分析领域的共性需求及其实现,而非仅针对单独的项目需求进行开发,有利于开发出有复用价值的构件.这部分形成了领域工程研究方向;

- 二是如何组织管理可复用的软件:对软件构件进行描述、分类、存储,提供简单且准确的检索机制.这部分形成了软件构件库研究方向;
- 三是如何组装已有构件形成新的系统:设计适当的构件模型,将软件代码封装成为可以单独使用的成分,基于构件模型的接口进行集成组装.这部分形成了应用工程研究方向.

代表性的基于构件软件复用方法是卡内基·梅隆大学(CMU)提出的软件产品线方法(software product line Conf.,简称 SPLC)和欧洲提出的产品家族工程(product family engineering).一个产品线是共享一组共同设计及标准的产品族,从市场角度来看,是在某市场片断中的一组相似的产品.产品线基于保证最大限度复用的产品族可复用构件而建立.产品线方法可以通过各种可复用软件构件,如需求、需求规约、构架、代码构件、文档、测试策略和计划、测试案例和数据、开发人员的知识和技能、过程、方法及工具等,支持系统化的软件复用.北京大学杨芙清教授提出的软件工业化生产模式及其对应的青鸟软件生产线系统则提出了完整的支持复用全过程的开发平台以及基于构件/构架的软件复用开发方法.

从 1990 年代中期开始,软件工程界出现了软件复用的研究和实践热潮.这一时期出现了两个专业的软件复用学术会议:Int'l Conf. on Software Reuse(ICSR)和 The Software Product Line Conf.(SPLC).在产业界,基于复用、基于构件以及产品线工程的实践成为主要软件企业的技术提升途径.

软件复用的基本单元从程序代码开始,发展到了面向对象的类以及封装的构件,包括运行态的构件(如 CORBA、EJB、DCOM 等分布式运行构件)一直在向大粒度方向进化,为的是复用的粒度大,效益更为明显.21 世纪开始的 Web 服务(Web service)及其面向服务的架构分析设计技术(service-oriented architecture,简称 SOA)提供了互联网上可访问的服务实体,这使得软件复用的研究和应用领域扩展到了互联网的软件实体中.针对 Web 服务的选择、服务质量(QoS)预测和服务组装的研究,成为软件复用领域的技术和应用新扩展.

5.2 基于开源软件的软件开发和复用

开源软件的发展,则为软件复用提供了更加广阔的空间.软件复用的关键因素在于要有大量可以复用的软件资源.开源软件的海量资源,成就了软件复用新的发展空间:一方面,信息检索技术的发展为查询合适的可复用软件提供了一种可行的机制;另一方面,程序语言及开发环境中支持复用的机制有所发展,如 Python 语言等能够更为容易地嵌入已有构件.开源软件的发展不仅仅是代码库的积累和开放,大量涉及开发技术细节的文献、信息也被开放在互联网上,更大程度地带来了开发社区和生态的新环境.在开发社区中,开发的需求讨论、缺陷修改的方案等都被邮件信息、缺陷报告工具、版本管理工具等记录.在开发生态系统中,Stackoverflow 这样的技术问答网站直接为用户回答技术细节问题.这些技术文献的搜集、组织和应用,形成了当前的软件复用的主要可复用资源池.

开源软件的开放源代码是复用的基本资源.不同于传统的基于构件的复用中争取代码封装以备组装使用的专门技术方案,开源软件的代码复用更多地是通过代码直接调用程序接口(API)来实现的,这对提供高质量、广泛代表性的使用样例(或使用示例)提出了更高的要求,从而也带来了开源软件中软件包关联过于复杂、庞大的系列问题.开源软件的另一主要复用机制则是代码框架的广泛使用,取代传统软件体系结构的专门定义,代码框架成为复用中的体系结构.在框架中增补相对应的软件代码,成为整体协同定制和发展的基础技术.以往的软件复用主要是针对企业局域、小规模的领域和组织,其数据内容、可复用资源数量有限,主要依托工程化方法实施.从开源软件的海量复用资源以及复用机制的转变来看,开源软件复用的核心问题已转变为在互联网广域环境下,面对软件大数据基础,如何高效实现大规模群体敏捷化开发的问题.

5.3 知识驱动的软件开发和复用

我们认为,知识驱动的软件开发方法(knowledge-driven software development,简称 KDSD)已成为当前软件复用的主要研究方向.可复用的软件实体仍然包括了代码片段、API、软件包、Web 服务、框架等软件基础资源.复用的核心关注点则转变为以软件知识为核心关注点,研究如何基于特定的知识结构以及认知方法和机制来描述、理解和利用可复用的软件实体.这其中,主要涉及的问题包括:

- 知识的表示:采用语义网络(semantic network)\知识图谱(knowledge graph)等技术来表达丰富的软件开发知识和领域知识;
- 知识的来源:可从软件代码获取高度结构化、精简、准确的领域知识体系,也可从丰富的软件相关信息中获取自然语言表达的知识;
- 知识的语义关联:建立软件代码知识和自然语言知识的关联,形成领域的语义模型.

在此基础上,软件开发工具能够以智能推荐的方式为开发人员提供帮助;更进一步地,可能部分实现基于自然语言需求描述自动生成对应的程序.当代的知识表示、信息检索和机器学习技术为这方面的发展提供了全新的技术途径.我们可以期望未来出现一种软件开发的智能化方法,尽可能准确地提供代码推荐和生成,从而使得软件复用方法的研究和实践从面向复用的对象转向基于知识驱动的智能化开发方法.

6 模型驱动的方法

如何高效、低成本地开发优质的软件产品,一直是计算机软件领域重点关注和研究的问题[68].伴随着信息化进程的迅猛发展,软件的复杂度在不断提高.同时,软件的演进形态也日益多样化[69].在这种情况下,如何有效地解决上述问题成为学术和工业界共同关注的焦点.模型驱动方法正是在这样的背景下逐渐受到重视,被认为是可以应对“高效、低成本地开发优质软件”的一条有效途径.而这一认识也伴随着实践不断得到深化,经历了从统一建模语言(unified modeling language,简称 UML)到模型驱动架构(model-driven architecture,简称 MDA)、从模型驱动架构到模型驱动工程(model-driven engineering,简称 MDE)、从模型驱动(model-driven)到基于模型(model-based)的过程.

6.1 从UML到MDA

在软件开发的长期摸索过程中,人们逐渐认识到,“提高解决问题的抽象层次”是有效利用抽象手段解决软件开发问题的一个非常具体而实用的途径(与软件技术发展密切相关的 3 个要素是计算机平台、人的思维模式和问题的基本特征[2].文中出现的“提高解决问题的抽象层次”中的“提高”指的就是向着更接近于人的思维模式方向上的提高).Mellor 在 21 世纪初曾指出[70]:过去的 50 多年里,人们利用“提高解决问题的抽象层次”处理软件开发的问题已经取得了两个较为显著的进展:(1) 开发出了具有较高抽象层次的程序设计语言;(2) 能够在更高抽象层次上实现软件复用.

然而这个发展道路也并非一帆风顺,在面向对象的语言、方法和技术逐渐成为主流的过程中,Booch、Jacobson 和 Rumbaugh 各自创建的 Booch、OOSE 和 OMT 方法都获得了工业界的支持,并形成了互不兼容的 3 种模型和方法体系.不同的工业应用在这 3 种不同的体系下形成了彼此之间难以理解和复用的软件制品,这让当时原本就不成熟的软件行业更是雪上加霜.如何形成一个能够支撑从需求到实现的软件行业标准,成为当时摆在软件发展道路上的一个亟需解决的问题.在这种情况下,3 位面向对象方法学的大师走到一起,融合了他们的 3 种软件开发方法和模型表示法,形成了统一建模语言(unified modeling language,简称 UML),通过非盈利的国际标准化组织对象管理组(object management group,简称 OMG)发布为建模规范,并最终在 2005 年被国际标准组织(Int’l Organization for Standardization,简称 ISO)采纳为工业标准.

由于 UML 设计者和 OMG 的共同影响力,UML 从 20 世纪 90 年代开始构造起就受到工业界的关注,在 1997 年发布 1.0 草案版时已经获得了工业界的普遍支持,并反过来引起学术界的研究热情,再历经 1.1~1.3 版本的演化,最终在 1.4 版本时成为 ISO 工业标准,更是奠定了 UML 在整个软件行业中的毋容置疑的通用建模标准的地位,这也形成了面向对象方法学发展道路上的一个重要的里程碑.

有了这样的技术积累,人们开始尝试在更高的抽象层次上开发软件,中间件的形成与发展也正是在这样的背景下逐步明晰和受到重视的.商家往往能够最早嗅探到技术发展背后的商机,并善加利用以推动自身的发展.于是,J2EE、.NET 等中间件技术开始各自占据市场,又仿佛当年 Booch、OOSE 和 OMT 方法之间的争斗.在这种情况下,OMG 再次提出了以模型为中心的软件开发框架性标准——模型驱动体系结构(model driven architecture,简称 MDA)[71],又一次得到了来自学术界和工业界的普遍关注.尽管 MDA 提出的直接动因是为了解

决异构中间件(middleware)平台的互操作障碍问题[71],但是由它所倡导的以模型为中心进行软件开发的思想很快得到了广泛支持,迅速成为研究热点.MDA 整合了 OMG 在建模语言、模型存储以及模型交换等方面的一系列标准,形成了一套基于模型技术的软件系统开发方法和标准体系.

6.2 从MDA到MDE

随着 MDA 研究热潮的迅速兴起,模型驱动软件开发这个词语逐渐被越来越多的学者所使用.此间,与模型相关的不同字眼也不断出现在不同的学术机构和社区(community)中,如 model-driven、model-based、model-related、model-engineering 等等.2005 年,模型驱动软件开发领域最重要的年会之一——UML Series(Int'l Conf. on the Unified Modeling Language)正式更名为 MoDELS/UML Series(Int'l Conf. on Model Driven Engineering Languages and Systems),这开始引起了人们对模型驱动软件开发领域自身术语使用上的关注.2007 年,模型驱动工程 MDE 这个术语正式出现在第 29 届 ICSE 年会的 FoSE(the Future of Software Engineering)分会上[72],标志着 MDE 成为模型驱动软件开发领域的代名词.

模型驱动工程以模型为首要软件制品(software artifact),通过建模为问题域构造软件系统的业务模型(business model),然后依靠模型转换(model transformation)驱动软件开发,(半)自动地产生最终完备的应用程序[73].可见,MDE 下的一切工作都是围绕着模型展开的.模型对于模型驱动工程而言,就如同对象对于面向对象软件开发一样,是基本的开发元素,这也就是为什么称其为"模型驱动"的原因之一.模型驱动使软件开发的抽象层次进一步提高到模型这个层次.有学者认为:模型驱动的思想,或者说以模型为中心的软件开发思想,正是抽象层次提高到一定程度的一个自然而然的结果[74].

在 MDE 中,模型指的是为了分析或解决问题而对系统某些方面的抽象描述.从广义上讲,它涵盖了软件系统的任意部分[75],包括需求、设计、文档、代码、部署、配置和测试用例等.代码也可以看作软件系统的模型,这是因为代码反映了软件程序被执行时的行为.然而,通常所指的 MDE 下的模型,更多的是狭义上的"设计草图"[76],即今天的开发模型(development models).如文献[77]中提到,模型是用来对问题域进行求解的手段;文献[78,79]认为:模型是针对实际问题或者业务逻辑构造,而抽象层次高于可执行代码或最终完备的应用程序的软件制品.

围绕着模型展开的一系列工作实际上也就构成了 MDE 中的各项研究工作.2007 年的国际软工大会上,France 等人[72]将目前这些工作中最重要和亟待解决的问题归结为如下几类挑战.

- 建模语言的挑战:这一类挑战主要来自于如何使建模语言支持在问题层面上进行建模,以及如何对模型进行严格的分析.
- 关注点划分的挑战:这一类挑战来源于对同一系统进行多视图建模的结果.由于不同侧面的模型可能是基于异构的语言开发的,而模型间又存在相互重叠的部分,因而很容易出现不一致等问题.
- 对模型操纵和管理的挑战:这一类挑战包括:(1) 对模型转换的定义、分析和使用;(2) 维护模型元素间的可追溯性,以支持模型演化(model evolution)和双向工程(roundtrip engineering);(3) 维护不同视点(viewpoint)之间的一致性;(4) 版本追踪;(5) 在运行期使用模型.

其中,前两类是针对建模的问题,最后一类主要是转换的问题.可以看出,模型驱动思想的核心就是建模和模型转换,它们也是完成 MDE 开发的两个最主要的过程.并且,建模过程和模型转换过程又是相互交织的,建模过程涵盖了一系列不同抽象层次模型的构造过程,而相应的转换过程也包括了不同抽象层次间以及相同抽象层次上的转换.

6.3 从Model-Driven到Model-Based

近年来,随着 MoDELS 国际会议影响力的不断扩大,软工领域的各个研究分支也在不断将工作发表在该会议上,这在很大程度上推动了整个软工领域对"模型"更为深入的思考,最开始的那种以"模型"为中心来驱动软件开发的想法也慢慢演化为如何将模型作为有效的抽象手段来刻画软件的某些重要性质,从而更好地支撑在不同研究领域的应用,如测试、验证、推荐、复用等等.在这种背景下,"Model-Driven"的含义慢慢向"Model-

Based”靠近.

模型驱动(model-driven)的术语主要是从 MDA 中产生出来,其最初的含义是强调系统的构造是通过以模型为中心来完成的,也就是说,模型的主要用来“驱动”软件从需求开始向实现过渡的这个过程.而随着 MDE 研究领域的逐渐形成,模型的价值在整个软工更大范围内得到关注,模型的概念也从最早的 UML 这个具象的对应体,慢慢回归到作为软件系统的“抽象描述”这样的本质上来.因而,基于模型(model-based)这个说法就成为更能够表达其宽泛含义的术语,在 MDE 领域得到了更多的应用,如基于模型的软件开发(model-based software development)、基于模型的测试(model-based testing)、基于模型的规约(model-based specification)等等.

这样一来,大大拓宽了 MDE 研究领域的范围,也极大地增强了软件工程各个研究分支与 MDE 的融合.其中,在最近几年比较受关注的研究热点有模型对运行时的支持,模型资产库的构造与应用,模型与人工智能、机器学习的结合以及更加宽泛的建模 IDE 的研究等.

- 模型对运行时的支持主要是通过预先定义或者动态获得的系统行为模型,对软件系统在运行时的行为进行监控,从而预判甚至是干预和调整系统动态行为;
- 模型资产库的构造与应用是近来备受关注的工作,特别是随着互联网上开源软件的急剧增加,可以被直接获得的软件资产已经形成了相当的规模.如何有效地使用这些软件资产,引起了学术界和工业界的格外关注.模型作为软件的抽象描述,可以从不同的视角和维度展示软件的语义信息,进而形成海量软件资产的“索引”,极大地提高了对开源软件的理解和复用能力;
- 模型与人工智能、机器学习结合的研究,是最近人工智能和机器学习热潮下的一个趋势.虽然具体的研究内容还没有得到普遍共识,但总的来讲,一方面是研究基于模型的人工智能和机器学习,另一方面是将人工智能和机器学习技术用到 MDE 研究中.目前来看,后者的研究工作更多一些.

建模 IDE 的研究一直都是 MDE 领域的一个重要研究方向,而且具有很好的实用价值.但与之前研究工作不同的是,目前建模 IDE 的研究不仅仅是 UML 家族或者 MDA 框架下的模型概念,更加泛化的模型是目前的一个趋势,甚至是手绘草图、形式化模型等都被逐渐纳入到所讨论的范畴中.

7 服务化方法

随着以互联网为主干,电信网、移动网、传感网等多种网络正在不断渗透融合,软件系统的运行环境正在逐步从静态、封闭、固定的单机环境转变为动态、开放、多变的网络环境.为了应对网络环境中各类分布式资源的共享和集成对计算机软件技术的挑战,软件服务得到了学术界和工业界的广泛关注[80].

7.1 基本思想

软件服务是指将软件的功能以服务的形式通过互联网来交付,可以被使用者(最终用户或者第三方客户端程序)直接使用的独立的基本单元.就其形态而言,软件服务一般基于可共享和集成的应用系统和资源来构建,对外则表现为一组相对独立的业务功能单元(通常是可供外部直接调用的应用编程接口,即 API),更加方便使用者直接使用.软件服务的一个重要目标是屏蔽开放网络环境带来的异构性问题,因而一般具有较高的抽象级别和独立性.这种独立性也带来了软件服务之间更加松散的耦合关系,从而使得使用者可以灵活选择服务并进行组装来生成增值服务.特别地,由于开放环境下存在大量类似的资源,使用者可以在量大面广、推陈出新的多个相同或相似服务中不断选择最合适的服务.

目前,软件服务不仅成为网络环境下最为重要的一种软件形态,也改变了传统的软件交付方式,使得软件产业开始从“以产品为中心的制造业”向“以用户为中心的服务业”转变.软件由服务的运营商负责运维,用户无需拥有软件的本地拷贝,也不必考虑软件的维护和升级,还可以按需定制和购买软件.软件服务使其使用者能够在不拥有软件产品的前提下,直接使用软件的功能,从而实现“即拿即用”的效果[81].这种“只求使用,不求拥有”的特性,也只有通过互联网和软件的组合才可能实现.进一步地,软件服务的使用者可以更加灵活地进行服务组装以生成增值服务,并能不断地选择最优服务来满足适应性的需求.

7.2 主要角色和开发过程

一般而言,基于服务的软件开发主要包含 3 类主要角色,即服务提供者、服务使用者和服务代理.其中,

- 服务提供者按照服务契约(service contract)实现了提供业务功能的软件模块.从业务角度看,它是服务的拥有者;从体系结构看,它是访问服务的平台.对于提供同一业务功能的服务,可以有多种不同的服务提供者;
- 服务使用者(即服务的用户)调用服务提供者所实现的服务,以完成特定的业务需求.从业务角度看,它是请求特定功能的业务;从系统体系看,它是寻找并调用服务或启动与服务交互的应用.服务请求者可以是用户通过客户端应用程序(如浏览器或智能手机)实现,也可以是没有用户界面的程序实现(如另一个服务);
- 服务代理是一个可搜索的第三方注册机构,如 UDDI 等.服务提供者将服务描述发布给服务代理,服务请求者在服务代理处查找服务并进行绑定(可分为静态绑定和动态绑定)和调用.

某种程度上看,软件服务可以看成是软件构件在网络环境下的进一步延伸和发展.围绕软件服务的开发过程可以分为如下几个[82].

- 服务封装和生成.服务提供者将软件系统中需要开放的功能和数据按照一定的标准(如 Web 服务)进行封装并发布.服务的生成一般是对已有的软件系统的功能和数据进行封装,也可以是服务提供者新开发的功能和数据;
- 服务查找和选择.服务使用者查找并选择满足其功能和非功能需求的服务.服务查找既可以通过统一的服务代理(如 UDDI),也可以通过搜索引擎来进行;
- 服务组装.服务使用者对若干服务进行组装来形成新的增值服务应用.组装的过程既可以按照预先定义的规则(如业务流程规约 WS-BPEL 或 WSCI)及其引擎自动/半自动地完成,也可以根据当前的情境以人机协同的方式来完成(如 Web Mashups);
- 服务演化.由于每个软件服务是相对独立的实体,其功能和质量均可能发生演化.相应地,通过服务组装而成的应用也会随之发生演化,可表现为服务的升级/降级、增加、退出和替换等.

7.3 发展过程

- 面向服务的体系结构和 Web 服务.

软件服务的思想可以追溯到 20 世纪 90 年代中期出现的“应用服务提供商(application service provider,简称 ASP)”.21 世纪的前 10 年,软件服务开始成为学术界和工业界的研究实践热点,其代表为面向服务的体系结构(service oriented architecture,简称 SOA)及其主要的实现技术——Web 服务的兴起[82].围绕 Web 服务互操作协议、发现、组装、语义、管理和质量保障等问题,已经有了大量的工作.从实际应用情况看,在互联网上,基于 SOAP(简单对象访问协议)的 Web 服务数量还比较少,且大多只提供相对简单的数据查询功能(如天气、股票查询等),Web 服务之间也很少构成复杂的业务逻辑.这里既有 Web 服务技术体系存在自身缺陷的原因,如广域网中基于 SOAP 协议的消息传输性能较差、服务质量难以保障等,也有一些非技术的因素,如 Web 服务相关的技术标准过于繁杂难以统一、面向互联网的服务注册中心(UDDI)难以管理和维护等.

- RESTful 服务.

2004 年前后,随着 Web 2.0 的兴起,对数据/内容开放共享、多方协同、用户交互体验的需求不断增加,越来越多的 Web 系统所有者均开始关注如何将自身的功能和数据以服务的形式开放出来.由于传统的基于 SOAP 的 Web 服务在互联网环境下难以进行规模化应用,于是,基于 REST(representational state transfer)体系结构风格的软件服务技术得到了广泛关注和使用.REST 由 Roy Fielding(Roy Fielding 是 HTTP 1.1 规范的起草者和 Apache Web 服务器的开发者,并且倡导创建了 Apache 开源软件基金会)于 2000 年左右提出[83],是 WWW 架构最为核心的应用软件体系结构风格.与 SOAP 相比,REST 体系结构风格提供了一种相对轻量的应用协议,通过描述资源状态变化的方式而不是远程过程调用(remote procedure call)的方式来使用软件服务,更加适合万维网

环境.特别地, 2007 年起,云计算开始成为互联网环境下最为重要的应用模式[84],基于 REST 体系结构风格的软件服务成为云计算的基础使能技术之一.几乎所有的主流软件服务提供商(主要是互联网公司,如谷歌、亚马逊、Facebook、Salesforce.com、百度、腾讯等),都开始采用遵循 REST 的 Web API 来发布软件服务供用户在线使用[85].此外,基于 REST 的 Web 服务功能也不仅限于简单的数据查询,应用的业务范围更广,如谷歌的 Gmail 服务、Salesforce.com 的 CRM(客户关系管理)服务和亚马逊的存储服务等,都是基于 REST 的 Web 服务.当然,在云计算环境下,软件服务仍有很多问题需要解决,包括对“黑盒”形态的遗产系统“解耦”和“安全”的服务化封装、服务的交付与展现、服务动态组装、服务的跨域安全性、服务资源管理等.

- 微服务和开发运行一体化.

近年来,“微服务”得到软件服务研究实践者的广泛关注[86].微服务是一种基于一组独立部署运行的小型服务来构建应用的方法.与传统的面向服务体系结构 SOA 应用相比,这些小型服务主要围绕应用系统业务能力来构建,采用尽量去中心化的机制管理,使用不同技术栈开发,通过轻量级通信机制交互.围绕微服务架构,一些新的研究正在不断涌现,如面向主流编程语言并支持微服务架构的软件开发框架、事件驱动的开发运行一体化(如 DevOps)基础设施架构[87]、细粒度基础资源及其状态监控、在线测试和快速部署、容错等关键技术等.

8 展望:面向人机物融合应用的软件开发方法

纵观软件开发方法和技术的发展历史,新方法与技术体系的诞生总是由软件基础支撑平台和相应的应用需求的重大变化来驱动的.宏观来说,软件开发就是给定待解的应用问题,由软件开发者通过智力活动过程构造出能够在所提供的平台上有效运行的、能够解决问题的软件.因而,软件开发方法的 3 个宏观要素可抽象概括为平台空间、认知空间、问题空间.软件方法与技术体系的发展过程就是在平台和需求变化驱动力的推动之下,对这 3 个空间的认识不断深化并在其间有效协同的过程[88],从而帮助驾驭软件开发的复杂性.即:尽量避免引入附属性的复杂性,更好地理解和应对本质性的复杂性.例如:前文所述的结构化方法乃是由于 20 世纪 70 年代计算机基础能力(计算、存储与外设)的快速发展和软件危机的出现而导致人们对基础的程序设计方法与语言的科学思考而产生的,它较好地协同了软件开发的平台空间与认知空间;面向对象方法与技术体系则进一步发展了从宏观角度控制复杂性的手段,如关注分离、信息隐蔽、模块化等,并强调将问题空间纳入软件设计的范畴,提出了与问题结构具有良好对应关系的面向对象程序模型的概念与支撑机制,从而协调了软件设计平台空间、认知空间以及问题空间[88].然而,经典的结构化和对象化方法与技术体系诞生于静态、封闭、可控的环境,难以完全适应像 Internet 这样开放、动态与难控的环境和多变的用户需求,为此,需要探索新的软件方法与技术体系.

8.1 面向Internet的软件方法与技术探索

21 世纪以来,Internet 的快速发展与普及为软件方法技术的发展带来新的驱动力:其基础平台从 Mainframe 经过网络化、微型化、并行化而逐步形成一个所谓的 Global Ubiquitous Computer[89];其应用方式从经典封闭应用模式经过资源化、普适化与服务化,逐步形成了以 *X*-Computing(grid computing,cloud computing,service computing,pervasive computing 等)、*X*-Systems(embedded system,hybrid system,cyber-physical system 等)和 *X*-Data(very large scale data,massive data,big data)为代表的多样化开放式应用模式.于是,计算平台空间已经从单个或多个可控计算机向开放的 Internet 平台发展,其主要作用已开始从“计算为主”逐步向“通信连接为主”的方面转变;认知空间已开始从“面向个体程序员”开始向“群体化和服务化方式”过渡,其关注点在面向对象的“平台⇒程序员⇒问题”的基础上,开始向关注“个体⇒群体开发者⇒大量使用者”的方面过渡;问题空间已从确定环境下的单个问题求解,到开放开发环境下的群体问题求解,开始向非确定环境下如何为大量最终用户提供优质服务的方面发展,其关注点已从求解问题转变到开放环境下资源的共享与协同[88].

为此,人们从不同角度对面向 Internet 的软件方法和技术[90]进行了探索,例如自适应软件系统[91]、面向 Agent 的软件工程[92]、面向 IoT 的程序设计[93]、自组织系统[94]等等.而网构软件是中国学者提出的一种面向 Internet 的新软件范型[95−97].网构软件范型针对前述 Internet 计算平台和应用模式的特点,强调软件的自治性,即软件系统由分布于网络中自主独立的实体构成;协同性,即这些自治实体通过动态联盟协作共同达成应用目标;

适应性,即软件系统自主感知环境和需求变化并主动适应之;演化性,即软件系统的结构和行为能够动态演化从而长期生存;激发性(emergent),即软件系统可能呈现由自主实体交互而激发的非预设行为;以及可信性,即以应用目标和用户体验为导向的软件质量保障.为支持网构软件范型,研究者从软件的架构模型、构造方法、运行支撑以及质量保障等方面展开了一系列研究.例如,在架构模型方面提出了基于动态体系结构的模型[98]和环境驱动的模型[99]等;在构造方法方面,提出了以体系结构为中心方法[100]和基于环境本体的能力规约[101]等;在运行支撑方面,提出了基于双向转换的运行时体系结构技术、动态软件更新技术等;在质量保障方面,研究了服务化网构软件的时序性质的规约与监控[102]、特征驱动的需求依赖分析[103]等.

8.2 新型软件开发方法与技术展望

当前,Internet 网络互联不断拓展和深化,它已从计算机的网络拓展到无处不在的物理世界中的各种传感和执行设备.同时,它也深入支撑并改造了人类用户的各种社会关系,形成一种人机物一体融合的综合发展态势.这或许孕育着对软件开发方法和技术体系的革命性挑战和机遇.

人机物融合中的所谓“机”,主要是指软件的基础平台支撑,它经历了从计算机经过计算平台到计算思维的转变.在此过程中,计算机开始从以工具为主要形态演变为与机器形态无关的平台概念,然后上升为计算思维的概念,变为人类除理论与实验之外认识世界的第 3 种手段.所谓“物”,主要是指面向现实世界的信息化空间.在宏观上,它经历了从个体化空间经过社会化空间逐步向自然化空间的转变,预示信息化技术不断从个体化机构化的应用逐步向人类社会与自然界的拓展与深入.所谓“人”,即是信息化社会的主要载体,经历了从信息化的工作人与娱乐人经过网络支持的虚拟社会人到各类信息及时支撑的自然化超人的转变,其在自然界与人类社会的能力越来越强,但其对信息技术的依赖也越来越大.

如果从前述(平台空间、认知空间、问题空间)三要素的观点看,这三者大致对应于上述机、人和物.但这里的平台空间也包括了人和物在计算思维下的抽象.要支持这种抽象,构造新型软件所运行的平台,必须要有新的技术手段.软件定义(software-defined,简称 SD)[104]的思路和技术给出了一种可能的途径.这里的问题空间与传统软件应用有很大的不同:传统软件只是负责现实世界业务过程中的信息处理环节,而现在的软件日益成为信息化了的现实世界中的应用的价值观的主要载体.这里,认知空间的主要问题就成为如何支持所谓自然化超人.这种以人为本的软件构造需要不同于传统的基于规约的软件构造,需要新的方法和技术的支持.

人机物三者的综合与协同发展是信息技术进步的主题,也是推动软件方法与技术不断发展的动力.例如,(人:信息化工作人,机:Wintel 模式,物:信息化工作空间,理念:人人一台 PC)、(人:被信息化服务人,机:Google 模式,物:信息化服务空间,理念:人人可以使用全世界的信息)、(人:虚拟社会人,机:Web As a Platform,物:虚拟化社会空间,理念:我为人人与人人为我)等均代表了信息技术发展的不同模式与发展阶段.在此意义下,当前正在发展与追求的主要目标是(人:自然化与信息化结合的超人,机:物联网/云计算/大数据/CPS,物:智慧校园/智慧城市/智慧地球,理念:Anytime,Anywhere,AnyService).在这样一个发展过程中,将推动软件方法与技术体系逐步从封闭独立软件系统到开放协同软件系统,从灵活好用的软件系统到智能可信软件系统的重大转型,进而推动软件方法与技术体系从“经典的面向对象软件方法与技术体系”经过“面向 Internet 软件方法与技术体系”逐步向“面向人机物融合的软件方法与技术体系”的跨越.

References:

[1] Zhang XX, Xu JF, *et al*., eds. Encyclopaediaof Computer Science and Technology. 3rd ed., Beijing: Tsinghua University Press, 2018 (in Chinese).

[2] Simon HA. The Sciences of the Artificial. 3rd ed., Cambridge: The MIT Press, 1996.

[3] Jr Brooks FP. The Mythical Man-Month. Boston: Addison-Wesley, 1995.

[4] Wirth N. A brief history of software engineering. IEEE Annals of the History of Computing, 2008,30(3):32−39.

[5] Jr Brooks FP. No silver bullet—Essence and accidents of software engineering. IEEE Computer, 1987,20(4):10−19.

[6] Dijkstra EW. Go to statement considered harmful. Communications of the ACM, 1968,11(3):147−148.

[7] Booch G. Object-Oriented Analysis and Design with Applications. 2nd ed., Redwood City: Benjamin Cummings, 1993.

[8] Meyer B. Object-Oriented Software Construction. Prentice Hall, 1988.

[9] Xu JF, Wang ZJ, Zhai CX. Object-Oriented Programming Language. Nanjing: Nanjing University Prerss, 1993 (in Chinese).

[10] Martin RC. Agile Software Development: Principles, Patterns, and Practices. Prentice Hall, 2002.

[11] Liskov BH, Wing JM. A behavioral notion of subtyping. ACM Trans. on on Programming Languages and Systems, 1994,16(6): 1811−1841.

[12] Ole-Johan D, Bjørn M, Kristen N. Common Base Language. Norwegian Computing Center, 1970.

[13] Parnas DL. A technique for software module specification with examples. Communications of the ACM, 1972,15(5):330−336.

[14] Hoare CAR. Proof of correctness of data representations. Acta Informatica, 1972,4:271−281.

[15] Liskov B, Zilles S. Programming with abstract data types. In: Proc. of the ACM SIGPLAN Symp. on Very High Level Languages. Santa Monica, 1974. 50−59.

[16] Chen PP. The entity-relationship model: Toward a unified view of data. ACM Trans. on Database Systems, 1976,1(1):9−36.

[17] Kay AC. The early history of smalltalk. In: Proc. of the 2nd ACM SIGPLAN Conf. on History of Programming Languages (HOPL-II). New York: ACM Press, 1993. 69−95.

[18] Coad P, Yourdon E. Object-Oriented Analysis. Prentice-Hall, 1990.

[19] Coad P, Yourdon E.Object-Oriented Design. Prentice-Hall, 1992.

[20] Rumbaugh JR, *et al*. Object-Oriented Modeling and Design. Prentice-Hall, 1991.

[21] Jacobson I. Object-Oriented Software Engineering: A Use Case Driven Approach. New York: ACM Press, 1972.

[22] Jacobson I, Booch G, Rumbaugh JR. The Unified Software Development Process. Boston: Addison-Wesley Longman Publishing Co., Inc., 1999.

[23] Erich G, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[24] Mary S, David G. Software Architecture: Perspectives on an Emerging Discipline. Upper Saddle River: Prentice-Hall, 1996.

[25] Mylopoulos J, Chung L, Yu E. From object-oriented to goal-oriented requirements analysis. Communications of the ACM, 1999, 42(1):31−37.

[26] Lopes CIVD. A language framework for distributed programming [Ph.D. Thesis]. Boston: Northeastern University, 1997.

[27] Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, Loingtier JM, Irwin J. Aspect-Oriented programming. In: Proc. of the Object-Oriented Programming (ECOOP'97). LNCS 1241, Berlin, Heidelberg: Springer-Verlag, 1997. 220−242.

[28] Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold WG. An overview of AspectJ. In: Proc. of the 15th European Conf. on Object-Oriented Programming (ECOOP 2001). London: Springer-Verlag, 2001. 327−353.

[29] Coady Y, Kiczales G, Feeley M, Smolyn G. Using aspectc to improve the modularity of path-specific customization in operating system code. In: Proc. of the 8th European Software Engineering Conf. on Held Jointly with 9th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2001. 88−98.

[30] Spinczyk O, Lohmann D, Urban M. Advances in aop with aspectc++. In: Proc. of the Conf. on New Trends in Software Methodologies, Tools and Techniques. Amsterdam: IOS Press, 2005. 33−53.

[31] Sampaio A, Rashid A, Chitchyan R, Rayson P. Ea-Miner: Towards automation in aspect-oriented requirements engineering. Trans. on Aspect-Oriented Software Development III. LNCS 4620, Berlin, Heidelberg: Springer-Verlag, 2007. 4−39.

[32] Shepherd D, Tourwé T, Pollock L. Using language clues to discover crosscutting concerns. In: Proc. of the Workshop on Modeling and Analysis of Concerns in Software (MACS 2005). New York: ACM Press, 2005. 1−6.

[33] Whittle J, Araujo J. Scenario modelling with aspects. IEEE Software, 2004,151(4):157−171.

[34] Dausend M, Raschke A. Towards a comprehensive extension of abstract state machines for aspect-oriented specification. Science of Computer Programming, 2016,131:22−41.

[35] Jacobson I, Ng PW. Aspect-Oriented Software Development with Use Cases. Addison-Wesley Professional, 2005.

[36] Tkatchenko M, Kiczales G. Uniform support for modeling crosscutting structure. In: Proc. of the Model Driven Engineering Languages and Systems. LNCS 3713, Berlin, Heidelberg: Springer-Verlag, 2005. 508−521.

[37] Aldawud T, Bader A. UML profile for aspect-oriented software development. In: Proc. of the 3rd Int'l Workshop on Aspect Oriented Modeling. 2003.

[38] Supakkul S, Chung L. A UML profile for goal-oriented and use case-driven representation of nfrs and FRS. In: Proc. of the 3rd ACIS Int'l Conf. on Software Engineering Research, Management and Applications (SERA 2005). Washington: IEEE Computer Society, 2005. 112−121.

[39] Chavez C, Lucena C. A meta-model for aspect-oriented modeling. In: Proc. of the Workshop on Aspect-Oriented Modeling with UML. 2002.

[40] Wimmer M, Schauerhuber A, Kappel G, Retschitzegger W, Schwinger W, Kapsammer E. A survey on UML-based aspect-oriented design modeling. ACM Computing Surveys (CSUR) Surveys, 2011,43(4):28.

[41] Xie T, Zhao J, Marinov D, Notkin D. Automated test generation for AspectJ programs. In: Proc. of the 1st Workshop on Testing Aspect-Oriented Programs (WTAOP 2005). Chicago, 2005.

[42] Zhao J. Data-Flow-Based unit testing of aspect-oriented programs. In: Proc. of the 27th Annual Int'l Conf. on Computer Software and Applications (COMPSAC 2003). Washington: IEEE Computer Society, 2003. 188−197.

[43] Massicotte P, Badri L, Badri M. Towards a tool supporting integration testing of aspect-oriented programs. Journal of Object Technology, 2007,6(1):67−89.

[44] Zhao J, Xie T, Li N, Towards regression test selection for AspectJ programs. In: Proc. of the 2nd Workshop on Testing Aspect-Oriented Programs (WTAOP 2006). New York: ACM Press, 2006. 21−26.

[45] Xu G, Rountev A. Regression test selection for AspectJ software. In: Proc. of the 29th Int'l Conf. on Software Engineering (ICSE 2007). Washington: IEEE Computer Society, 2007. 65−74.

[46] Xie T, Zhao J, Marinov D, Notkin D. Detecting redundant unit tests for AspectJ programs. In: Proc. of the 17th Int'l Symp. on Software Reliability Engineering. 2006. 179−190.

[47] Harman M, Islam F, Xie T, Wrappler S. Automated test data generation for aspect-oriented programs. In: Proc. of the 8th Int'l Conf. on Aspect-Oriented Software Development. 2009. 185−196.

[48] Parizi RM, Ghani AAA, Lee SP, Khan SUR. RAMBUTANS: Automatic AOP-specific test generation tool. Int'l Journal of Software Tools and Technology Transfer, 2017,19:743−761.

[49] Xu D, Ariss OE, Xu W, Wang L. Aspect-Oriented modeling and verification with finite state machines. Journal of Computer Science and Technology, 2009,24(5):949−961.

[50] Ubayashi N, Tamai T. Aspect-Oriented programming with model checking. In: Proc. of the 1st Int'l Conf. on Aspect-Oriented Software Development (AOSD 2002). New York: ACM Press, 2002. 148−154.

[51] Binkley D, Ceccato M, Harman M, Ricca F, Tonella P. Automated refactoring of object-oriented code into aspects. In: Proc. of the 21st IEEE Int'l Conf. on Software Maintenance (ICSM 2005). Washington: IEEE Computer Society, 2005. 27−36.

[52] Breu S. Aspect mining for large systems. In: Proc. of the Companion to the 21st ACM SIGPLAN Symp. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2006). New York: ACM Press, 2006. 641−642.

[53] Marin M, Deursen A, Moonen L. Identifying crosscutting concerns using fan-in analysis. ACM Trans. on Software Engineering and Methodology, 2007,17(1):1−37.

[54] Badri M, Kout A, Badri L. Investigating the effect of aspect-oriented refactoring on the unit testing effort of classes: An empirical evaluation. Int'l Journal of Software Engineering and Knowledge Engineering, 2017,27(5):749−790.

[55] Mourad A, Laverdiere MA, Debbabi M. An aspect-oriented approach for the systematic security hardening of code. Computers & Security, 2008,27(3-4):101−114.

[56] Ali-Gombe A, Saltaformaggio B, Ramanujam J, Xu DY, Richard III GG. Toward a more dependable hybrid analysis of android malware using aspect-oriented programming. Computers & Security, 2018,73:235−248.

[57] Pereira RH, Pérez-Schofield BG, Ortin F. Modularizing application and database evolution—An aspect-oriented framework for orthogonal persistence. Software Practice and Experience, 2017,47(2):193−221.

[58] Soares S, Laureano E, Borba P. Implementing distribution and persistence aspects with AspectJ. In: Proc. of the 17th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002). New York: ACM Press, 2002. 174−190.

[59] Fan GS, Yu HQ, Chen LQ. A formal aspect-oriented method for modeling and analyzing adaptive resource scheduling in cloud computing. IEEE Trans. on Network and Service Management, 2016,13(2):281−294.

[60] Bodden E, Havelund K. Aspect-Oriented race detection in Java. IEEE Trans. on Software Engineering, 2010,36(4):509−527.

[61] Chen XQ, Yang FQ. Research on aspect oriented operating systems. Ruan Jian Xue Bao/Journal of Software, 2006,17(3):620−627 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/620.htm [doi: 10.1360/jos170620]

[62] Mei H, Cao DG. ABC-S2C: Enabling separation of crosscutting concerns in component-based software development. Chinese Journal of Computers, 2005,28(12):2036−2044 (in Chinese with English abstract).

[63] Cui ZQ, Wang LZ, Liu HG, Li XD. Computational error handling as aspects: A case study and evaluation. Ruan Jian Xue Bao/ Journal of Software, 2011,22(11):2639−2651 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3892.htm [doi: 10.3724/SP.J.1001.2011.03892]

[64] Mcllroy D. Mass produced software components. In: Proc. of the NATO Conf. on Software Engineering. 1968. 138−155.

[65] Yang FQ, Mei H, Li KQ. Software reuse and software component technology. ACTA ELECTRONICA SINICA, 1999,27(2):68−75 (in Chinese with English abstract).

[66] Mili H, Mili F, Mili A. Reusing software: Issues and research directions. IEEE Trans. on Software Engineering, 1995,21(6): 528−562.

[67] Zhang XX. Encyclopaediaof Computer Science and Technology. 2nd ed., Beijing: Tsinghua Unversity Press, 2005 (in Chinese).

[68] Pfleeger SL. Software Engineering: Theory and Practice. 2nd ed., Upper Saddle River: Prentice Hall, 2001.

[69] Yang FQ, Mei H, Lü J, Jin Z. Some discussion on the development of software technology. ACTA ELECTRONICA SINICA, 2002,30(12A):1901−1906 (in Chinese with English abstract).

[70] Mellor SJ, Scott K, Uhl A, Weise D. MDA Distilled: Principles of Model-Driven Architecture. Addison Wesley, 2004.

[71] Object Management Group, Miller J, Mukerji J, eds. MDA Guide Version 1.0.1. OMG White Paper, 2003.

[72] France R, Rumpe B. Model-Driven development of complex software: A research roadmap. In: Proc. of the Future of Software Engineering (FoSE) on the 29th Int'l Conf. on Software Engineering (ICSE 2007). 2007. 37−54.

[73] Schmidt DC. Guest editor's introduction: Model-driven engineering. IEEE Computer, 2006,39:25−31.

[74] Frankel DS. Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley, 2003.

[75] Bézivin J. On the unification power of models. Software and Systems Modeling (SoSyM), 2005,4:171−188.

[76] Kurtev I, Bézivin J, Jouault F, Valduriez P. Model-Based DSL frameworks. In: Proc. of the ACM SIGPLAN Int'l Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006). ACM Press, 2006.

[77] Brown A, Conallen J, Tropeano D. Introduction: Models, modeling, and model-driven architecture (MDA). In: Proc. of the Model-Driven Software Development. Springer-Verlag, 2005. 1−16.

[78] Liu J, He JF, Miao HK. A strategy for model construction and integration in MDA. Ruan Jian Xue Bao/Journal of Software, 2006,17(6):1411−1422 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/20060616.htm [doi: 10.1360/jos 171411]

[79] Mellor SJ, Balcer MJ. Executable UML: A Foundation for Model-Driven Architecture. Addison-Wesley Professional, 2002.

[80] Jiang ZM. Development of IT industry in China in the new age. Journal of Shanghai Jiaotong University, 2008,42(10):1589−1607 (in Chinese with English abstract).

[81] Mei H, Liu XZ. Software technology in the Internet era: Overview and outlook. Chinese Science Bulletin, 2010,55(13):1214−1220 (in Chinese).

[82] Papazoglou MP, Georgakapoulos G. Service-Oriented computing. Communications of the ACM, 2003,46(10).

[83] Fielding RT, Taylor RN. Principled design of the modern Web architecture. In: Proc. of the Int'l Conf. on Software Engineering (ICSE 2000). 2000. 407−416.

[84] Vinoski S. REST eye for the SOA guy. IEEE Internet Computing, 2007,11(1):82−84.

[85] Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. A view of cloud computing. Communications of the ACM, 2010,53(4):50−58.

[86] Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Software, 2016,33(3):42−52.

[87] Zhu LM, Bass L, Champlin-Scharff G. DevOps and its practices. IEEE Software, 2016,33(3):32−34.

[88] Lü J, Ma XX, Tao XP, Xu F, Hu H. Researches and progresses on Internetware. Science in China (Series E), 2006,36(10): 1037−1080 (in Chinese with English abstract).

[89] Milner R. Theories for the global ubiquitous computer. In: Proc. of the Foundations of Software Science and Computation Structures. LNCS 2987, Berlin: Springer-Verlag, 2004.

[90] Lü J, Rosenblum DS, Bultan T, *et al*. Roundtable: The future of software engineering for internet computing. IEEE Software, 2015, 32(1):91−97.

[91] Cheng BHC, *et al*. Software engineering for self-adaptive systems: A research roadmap. In: Proc. of the Software Engineering for Self-Adaptive Systems. LNCS 5525, 2009. 1−26.

[92] Zambonelli,F, Omicini A. Challenges and research directions in agent-oriented software engineering. Autonomous Agents and Multi-Agent Systems, 2004,9(3):253−283.

[93] Beal J, Pianini D, Viroli M. Aggregate programming for the Internet of Things. IEEE Computer, 2015,48(9):22−30.

[94] Zambonelli F, Viroli M. A survey on nature-inspired metaphors for pervasive service ecosystems. Journal of Pervasive Computing and Communications, 2011,7:186−204.

[95] Mei H, Lü J. Internetware: A New Paradigm for Internet Computing. Springer-Verlag, 2016.

[96] Yang F, Lü J, Mei H. Technical framework for Internetware: An architecture centric approach. Science in China (Series F: Information Sciences), 2008,51(6):610−622.

[97] Lü J, Ma XX, Huang Y, Cao C, Xu F. Internetware: A shift of software paradigm. In: Proc. of the 1st Asia-Pacific Symp. on Internetware (Internetware 2009). 2009.

[98] Mei H, Huang G, Lan L, Li J. A software architecture centric self-adaptation approach for Internetware. Science in China (Series F: Information Sciences), 2008,51(6):722−742.

[99] Lü J, Ma XX, Tao XP, Cao C, Huang Y, Yu P. On environment-driven software model for Internetware. Science in China (Series F: Information Sciences), 2008,51(6):683−721.

[100] Mei H, Huang G, Zhao H, Jiao W. A software architecture centric engineering approach for Internetware. Science in China (Series F: Information Sciences), 2006,49(6):702−730.

[101] Wang P, Jin Z, Liu L, Cai G. Building toward capability specifications of Web services based on an environment ontology. IEEE Trans. on Knowledge and Data Engineering (TKDE), 2008,20(4):547−561.

[102] Guoquan W, Jun W, Chunyang Y, Hua Z, Tao H, Hong H. Specification and monitoring of data-centric temporal properties for service-based systems. Journal of Systems and Software, 2012,85(12):2738−2754.

[103] Zhang W, Mei H, Zhao H. Feature-Driven requirement dependency analysis and high-level software design. Requirements Engineering (RE), 2006,11(3):205−220.

[104] Mei H, Guo Y. Toward ubiquitous operating systems: A software-defined perspective. Computer, 2018,51(1):50−56.

附中文参考文献:

[1] 张效祥,徐家福,等,编.计算机科学技术百科全书.第 3 版,北京:清华大学出版社,2018.

[9] 徐家福,王志坚,翟成祥.对象式程序设计语言.南京:南京大学出版社,1993.

[61] 陈向群,杨芙清.面向 Aspect 的操作系统研究.软件学报,2006,17(3):620−627. http://www.jos.org.cn/1000-9825/17/620.htm [doi: 10.1360/jos170620]

[62] 梅宏,曹东刚.ABC-S2C:一种面向贯穿特性的构件化软件关注点分离技术.计算机学报,2005,28(12):2036−2044.

[63] 崔展齐,王林章,刘慧根,李宣东.面向方面的误差处理技术:实例研究和评估.软件学报,2011,22(11):2639−2651. http://www.jos.org.cn/1000-9825/3892.htm [doi: 10.3724/SP.J.1001.2011.03892]

[65] 杨芙清,梅宏,李克勤.软件复用与软件构件技术.电子学报,1999,27(2):68−75.

[67] 张效祥.计算机科学技术百科全书.第 2 版,北京:清华大学出版社,2005.

[69] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展.电子学报,2002,30(12A):1901−1906.

[78] 刘静,何积丰,缪淮扣.模型驱动架构中模型构造与集成策略.软件学报,2006,17(6):1411−1422. http://www.jos.org.cn/1000-9825/20060616.htm [doi: 10.1360/jos171411]

[80] 江泽民.新时期我国信息技术产业的发展.上海交通大学学报,2008,42(10):1589−1607.

[81] 梅宏,刘譞哲.互联网时代的软件技术:现状与趋势.科学通报,2010,55(13):1214−1220.
[88] 吕建,马晓星,陶先平,徐锋,胡昊.网构软件的研究与进展.中国科学(E 辑),2006,36(10):1037−1080.

马晓星(1975－),男,江苏南通人,博士,教授,博士生导师,CCF 专业会员,主要研究领域为软件工程,自适应软件系统.

刘譞哲(1980－),男,博士,副教授,CCF 专业会员,主要研究领域为服务计算,Web 技术,软件工程.

谢 冰(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,计算机科学理论,分布式系统.

余萍(1979－),女,博士,副教授,CCF 专业会员,主要研究领域为软件工程,软件方法学,云计算,大数据技术.

张天(1978－),男,博士,副教授,CCF 高级会员,主要研究领域为模型驱动工程.

卜磊(1983－),男,博士,副教授,CCF 专业会员,主要研究领域为实时混成系统,信息物理融合系统,形式化验证与分析.

李宣东(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为软件工程,形式化验证,可信软件开发.