中国科学: 信息科学 2014 年 第 44 卷 第 6 期: 743–761
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
可再生能源互联网专题 · 论文 科学与技术前沿论坛

复杂软件系统的成长性构造与适应性演化

王怀民[①], 吴文峻[②], 毛新军[①], 丁博[①*], 郭长国[③], 李未[②]

① 国防科学技术大学计算机学院, 长沙 410073
② 北京航空航天大学计算机学院, 北京 100191
③ 中国电子设备系统工程公司, 北京 100039
* 通信作者. E-mail: dingbo@nudt.edu.cn

收稿日期: 2014–01–20; 接受日期: 2014–05–06

摘要 随着信息网络技术的渗透性发展, 复杂软件系统正在成为一种泛在的新型软件形态. 此类软件系统通常由相当数量的局部自治的软件系统相互耦合关联而成, 具有 “系统之系统”、“信息 — 物理” 融合系统和 “社会 — 技术” 交融系统的特点, 表现出成员异质、边界开放、行为涌现、持续演化等一系列新的性质. 这些特征打破了传统基于 “还原论” 思想的软件工程理论和技术所基于的基本假设, 使其难以适用于复杂软件系统的构建. 本文分析复杂软件系统的内涵、形成特征和基本性质, 深入讨论复杂软件系统在构造和演化环节所面临的挑战, 借鉴互联网以及生命系统、社会系统和经济系统等复杂系统的形成和演进模式, 提出复杂软件系统的 “成长性构造” 和 “适应性演化” 法则, 阐述这两条法则所涉及的主要科学问题和关键支撑技术. 本文试图为复杂软件系统的构建和发展提供新的方法学和架构层面的支持.

关键词 复杂软件系统 成长性构造 适应性演化 软件开发 软件演化

1 引言

近年来, 软件系统的复杂程度持续增长. 这种增长首先表现在软件的规模上. 例如, 互联网的前身 ARPANET 最初仅连接了 4 个节点, 而 2014 年 4 月支持即时通信的分布式应用 QQ 实现了超过 2 亿个节点同时在线 [1)], 其中超过 70% 是移动终端设备; IBM 正在英国建设智能能源云 (UK smart energy cloud) 项目 [2)], 计划在 2020 年将英国 5200 万台智能电表联接在一起, 形成能源互联网络; 美军正在建设的 “全球信息栅格”(global information grid) 旨在通过网络和分布计算软件实现从传感器到射手、从总统府到散兵坑的 “无缝信息链接”, 其规模超过 200 万个结点, 与此同时美军正在调研如何建设超过 10 亿行代码的软件系统 [1].

规模上的持续量变引起系统复杂性的质变. 在规模持续增长的过程中, 许多大型软件系统在内部结构、外部交互、演化方式等维度上也表现出了与传统分布计算软件迥异的特征, 它们兼具了 “系统

1) http://tech.qq.com/a/20140412/000129.htm.
2) https://www.ibm.com/smarterplanet/uk/en/smart_grid/article/smart_energy_cloud.html.

引用格式: 王怀民, 吴文峻, 毛新军, 等. 复杂软件系统的成长性构造与适应性演化. 中国科学: 信息科学, 2014, 44: 743–761, doi: 10.1360/N112014-00009

之系统”、“信息 — 物理” 融合系统和 “社会 — 技术” 交融系统的特点. 我们称此类软件系统为复杂软件系统, 其典型实例包括互联网软件系统、能源互联网软件系统、各类大型联合指控软件系统、国家和全球金融信息系统等. 它们在如下两个方面对现有软件开发和维护技术 [2] 提出了挑战:

(1) 在构造方面, 由于其规模巨大、内部结构复杂, 复杂软件系统不可能一次性设计、开发和部署. 在时间尺度上, 其持续开发、部署、更新和调整的过程可能长达数年甚至数十年. 从参与者的角度而言, 这一过程涉及到大量不同目标的投资者, 以及不同类型的开发者、维护者和使用者等大量利益相关者; 从系统成分的角度而言, 这一过程涉及到高度异构的各类自治系统的持续集成. 传统软件开发方法中所基于的需求事先可以精确获取、开发过程可以严格可控、运行效果可以预期等假设都不再成立, 经典的 “自上向下分解、自下而上组装” 的软件开发方法很难有效支撑这类复杂软件系统的构造.

(2) 在运行方面, 传统软件运行时的风险主要来自于软件本身的缺陷, 因此人们一直在追求通过测试、形式化验证等手段来 “防患于未然”. 然而, 复杂软件系统更多的风险来自于其固有的内在冲突, 以及与外部环境无法事先精确预期的各类交互. 例如, 文献 [3] 就以美国证券市场为例指出, 尽管其金融信息系统中各个局部自治软件系统 (如各个公司的高频交易系统) 本身可能并无缺陷, 但这些自治软件系统之间, 以及信息系统与社会系统、物理系统之间相互作用, 涌现出了一系列无法预料的行为, 近年来许多美股异常事件都可以归咎于此, 其中就包括了 2010 年 5 月 6 日导致道琼斯指数跌近千点、造成了巨大经济损失的 “闪电崩盘”(flash crash) 事件 3).

复杂软件系统将是未来软件系统的主流形态之一, 在社会、经济、军事活动中占据支撑性、基础性的地位. 以发展中的能源互联网软件系统为例, 未来成千上万位于不同地理位置的自治软件实体 (如发电设备的嵌入式监控软件、能源路由器的能源信息交换和决策软件、电力用户的各类耗电分配和度量软件等) 相互交联, 协同完成计算、通信、控制、测量、分析等功能, 是支撑能源互联网高效运转的数字中枢 [4]. 因此, 如何驱动复杂软件系统的构造, 并保证其能够持续可靠运转, 是软件领域研究者和实践者当前所面临的重大挑战.

本文针对复杂软件系统的特点, 提出以 “成长性构造” 和 “适应性演化” 为核心的复杂软件系统构造和演化法则, 进而对这一思路下的挑战和技术体系进行探讨, 以期能够为复杂软件系统的构造与演化提供方法学方面的支持. 本文后续章节内容如下: 第 2 部分给出复杂软件系统定义, 阐述其结构特征和基本性质; 第 3 部分详细分析这些特点和性质对传统软件开发技术所带来的挑战; 第 4 部分提出复杂软件系统的成长性构造和适应性演化法则, 概述其基本思想; 第 5 部分介绍复杂软件系统成长性构造技术体系; 第 6 部份阐述复杂软件系统的适应性演化技术体系; 第 7 部分给出了本文的结论.

2 复杂软件系统的特点和性质

复杂软件系统是一类具有显著复杂系统 [5] 特点的软件系统, 可以将其定义如下:

定义 1 (复杂软件系统) 复杂软件系统是指由大量局部自治软件系统持续集成、相互耦合关联而成的大型软件系统, 此类软件系统与其所作用的社会系统和物理系统密切相关, 系统要素之间的耦合交互关系动态变化且日趋复杂, 整个系统的行为难以通过各自治软件系统特征的简单叠加加以刻画.

2.1 复杂软件系统的结构特征

从定义 1 可以看出, 复杂软件系统的内部结构可以从 3 个角度描述: 首先, 复杂软件系统是由大

3) http://en.wikipedia.org/wiki/2010_Flash_Crash.

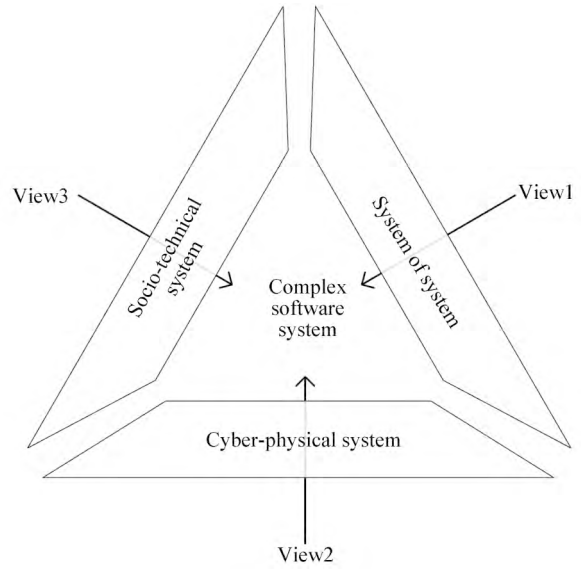

图 1 理解复杂软件系统的不同结构角

Figure 1 Different structure views to understand complex software system

量局部自治软件系统所耦合而成的 “系统之系统”; 其次, 它们不仅涉及信息系统, 而且与物理世界紧密联系、相互作用, 是典型的 “信息 — 物理” 融合系统 (cyber-physical system)[6]; 再次, 此类系统还是 “社会 — 技术” 交融的系统, 人、组织等社会要素与软件等技术要素密不可分. 以下将从这 3 个视图出发, 对复杂软件系统的结构特征进行分析 (图 1).

视角 1: 复杂软件系统是 “系统之系统”

在系统科学领域, “系统之系统 (systems of systems)”[7] 是指由一组既有系统所组成的系统, 这些既有系统本身具有操作独立性 (operational independence) 和管理独立性 (managerial independence). 著名学者钱学森也提出了类似的复杂巨系统 [8] 概念, 强调一个系统中有很多独立系统且关联关系十分复杂. 在传统软件系统中, 软件的各个子部件都是输入输出定义明确的功能模块, 本身往往并不具备操作独立性和管理独立性, 在架构上往往是 “点”(如独立的桌面办公软件) 或者 “线”(如 “表示 — 业务逻辑 — 数据库”3 层架构的分布式企业计算系统). 复杂软件系统则不然, 它是典型的系统之系统, 由相当数量并行工作、无法被统一集中控制的局部自治系统组成, 这些系统以松耦合的方式交互, 在内部结构维度上呈现出了 “网” 的特征: 局部自治系统种类繁多, 部署结点数成千上万甚至更多, 各类成分关系复杂多变, 存在多重宏观和微观层次, 以及不同层次之间的关联.

从软件的角度看, 互联网就是一种典型的复杂软件系统. 再看发展中的能源互联网, 无论是核心的大型发电厂和变电站, 还是小型绿色能源发电装置和能量贮存站, 都可以自由组网, 形成大大小小的能源自治网络, 并通过不同层次的能源路由器互通连接, 形成覆盖全球用户的能源网络. 相应地, 每一个能源自治网络上的软件系统都是一个局部自治系统, 可以独立运转, 具有自身的利益目标 (如维护自治网络上能源供需平衡), 需要首先实现独立运转和自身目标, 然后才会考虑接入到整个能源互联网软件大系统问题.

视角 2: 复杂软件系统是 “信息 — 物理” 融合系统

复杂软件系统不仅涉及信息系统, 而且与其所作用的物理世界密切相关. 作为典型的 “信息 — 物理” 融合系统, 复杂软件系统本身是物理过程和计算过程的有机融合和深度协作, 通过计算、通信和控

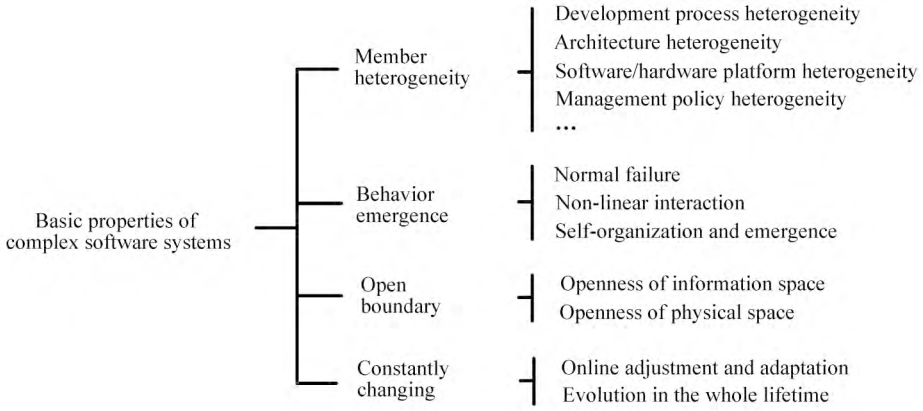

图 2　复杂软件系统的基本性质

Figure 2　Basic properties of complex software systems

制来增强物理实体的适应能力. 仍以能源互联网为例, 各类软件与能源互联网的能源采集和发生器、能源存储系统、能源输送和路由系统等物理设备之间发生着从若干微秒到数小时时间尺度的复杂交互模式, 是一个综合计算、网络和物理环境的多维复杂系统.

视角 3: 复杂软件系统是 “社会 — 技术” 交融系统

在与物理世界紧密结合的同时, 复杂软件系统与人、组织等社会因素也具有极为紧密的关系. 人 (社会组织) 和软件系统边界模糊化, 前者既是软件系统的输入和输出, 也在某种程度上充当着软件系统的 “元级” 成分, 通过与软件系统的密切交互影响着软件系统的行为模式和功能表达, 可以被视作系统的有机组成部份. 仍以能源互联网为例, 它也是与广泛用户和能源提供者存在着密切交互作用的社会系统. 这一系统涉及到社会方方面面的人员, 包括各种新能源的提供者、能源互联网的管理和维护者、以及广大的能源消费者. 在能源互联网的支撑下, 能源的提供者和用户面向不同的用能要求和场景, 可以构建和形成不同粒度和规模的能源交换市场, 而网络的管理者则依据市场发展的要求制订和执行能源交换的政策和规则, 促进能源市场向着资源优化、满足用户需求的方向演化和发展.

2.2　复杂软件系统的基本性质

作为 “系统之系统”、“信息 — 物理” 融合系统和 “社会 — 技术” 交融系统, 复杂软件系统表现出了如下一些迥异于传统软件系统的性质 (图 2).

(1) 成员异质性.　作为 “系统之系统”, 复杂软件系统各个局部自治系统或者是既有系统, 或者是新开发系统. 因此, 这些自治系统的开发过程、体系结构、软硬件平台、管理策略等往往是异构的, 所涉及的人、管理域等数量众多且可能存在利益冲突. 既有系统往往由不同的队伍开发, 具有不同的进度、过程和目标, 为不同的利益相关群体服务. 即使是新开发的系统, 也可能是按照不同的思路和方法设计, 在环境假设、数据一致性等方面存在差异.

(2) 行为涌现性.　一方面, 由于无法有效、精确地在宏观层面上进行集中控制和管理, 加上规模庞大、且持续发展变化, 各个局部自治软件系统层面上的失效在 “系统之系统” 中表现为常态. 另一方面, 在 “系统之系统” 中, 还存在非线性相互作用、自组织和整体行为涌现现象. 在复杂软件系统中, 整体的行为很难简单地通过各个局部自治系统行为来进行刻画, 机械的 “还原论” 在此类系统中不再成立. 系统层面上通过非线性作用涌现出的行为可能是有益的, 也可能是危害性的, 前文所给美国金融信息系统的例子就是一个典型实例.

(3) 边界开放性. 在复杂软件系统中, 一方面复杂软件系统所在的信息空间本身是开放的, 表现为系统运行往往需要依赖于大量外部软件实体, 软件自身的构成不具有明确、封闭的边界; 另一方面, 作为 "信息 — 物理" 融合系统和 "社会 — 技术" 交融系统, 物理世界和人类社会固有的开放性将体现在复杂软件系统中, 也使得复杂软件系统不可能成为一个封闭的实体.

(4) 持续变化性. 由于社会、技术、物理 3 个维度的紧密交融, 复杂软件系统也必然是持续变化的系统. 人类社会和物理世界存在固有的变化性, 将映射到复杂软件系统的技术维度上, 表现为软件需求和软件运行环境无法事先 "冻结" 并精确描述, 复杂软件系统需要在运行时朝着适应用户需求和环境变化的方向不断演进. 以能源互联网软件系统为例, 随着能源技术进步和用电需求的变化, 能源互联网软件需要将不断维护和更新; 各种能源设备随电力电子技术的革新而能力提升, 其相应的嵌入式控制软件和系统的骨干能源路由器的核心软件, 都需做相应的升级; 随着智能电表和能耗控制技术的普及, 住宅和企业用户的耗电行为推陈出新, 它们的耗电负载模式也将随着变化, 能源互联网软件需要重新构造和优化, 不断适应新的模式.

3 基于还原论的软件开发方法所面临的挑战

作为一门学科, 软件工程诞生于 20 世纪 60 年代末的 "软件危机". 今天, 随着复杂软件系统的出现, 研究者和实践者再一次面临了软件复杂性超出现有方法能力范围的困境, 迫切需要新的理论和技术来应对新一轮 "软件危机".

3.1 基于 "还原论" 的传统软件开发方法

在软件工程概念提出以前, 软件编程人员采用类似作坊式的方法来设计软件 [2], 缺乏系统的、有针对性的方法指导. 随着 20 世纪 60 年代计算机技术的快速发展, 软件系统的规模越来越大、复杂性越来越高, 软件开发的成功率越来越低、运行时的可靠性问题也越来越突出. 为了应对这一 "软件危机", 北大西洋公约组织 (NATO) 于 1968 年召开了首次软件工程学术会议, 指出 "软件开发应该是类似工程的活动"[9]. 在这一思路下, 软件工程领域提出了一系列理论、方法、技术和工具, 并在实践中得到了成功的应用.

20 世纪 60、70 年代, 软件工程主要关注结构化的软件开发技术和瀑布软件过程模型 [10]. 20 世纪 80 年代, 面向对象软件工程 [11] 得到了迅速发展, 它代表了一种新的软件范型, 通过封装、重用、高层抽象等手段控制复杂性, 提高了软件开发的效率和质量, 在一定程度上缓解了软件危机. 进入 20 世纪 90 年代, 在对象技术趋于成熟和持续推广应用的基础上, 构件 [12]、软件体系结构 [13]、软件中间件 [14]、软件过程管理 [15] 等技术得到了学术界和工业界的广泛关注和重视. 这些经典软件技术均基于 "还原论" 思想, 其应对软件复杂性的基本思路是 "分而治之", 采用的是自顶向下、计划驱动的模式: 在获得精确描述的用户需求之后, 在设计阶段自顶向下分解, 将软件需求分解简化为可管理和易开发的基本功能部件, 然后再通过自底向上静态组装来获得目标软件, 达到 "整体等于部份之和" 的效果 [16,17].

3.2 "还原论" 软件开发方法的局限性

如前所述, 主流、实用化的软件开发方法基于如下一些假设: 用户需求可以被事先完整分析、获取和描述; 软件开发过程精确可控; 软件具备明确边界和固定的结构; 软件运行所依赖的各类资源均

被妥善管理、集中调度. 然而, 复杂软件系统成员异质、行为涌现、边界开放、持续变化等性质将上述假设一一推翻, 在与软件开发活动密切相关的如下一些方面产生了本质差异, 如何有效构建此类系统已经超出经典软件开发方法关注的范畴.

(1) 开发活动目标. 在基于 “还原论” 的软件开发方法中, 衡量软件质量的标准是正确性, 即需求规格说明是否被完全满足. 而在复杂软件系统中, 需求和运行环境难以事先确定, 并且会持续变化, 因此衡量软件质量的标准应当是适应性, 即适应需求与环境的变化.

(2) 软件构造思路. 在基于 “还原论” 的软件开发方法中, 功能可以自顶向下线性分解, 产品可以通过机械的自下而上组装获得. 而在复杂软件系统中, 存在自组织行为和涌现现象, 系统行为往往是成员之间非线性相互作用的结果, 因此需要考虑既有系统再连接的构造方式.

(3) 对冲突与失效的观点. 在基于 “还原论” 的软件开发方法中, 成分间所有冲突都必须被解决, 缺陷能够被剔除, 失效应该极少产生. 而在复杂软件系统中, 冲突的产生是固有的, 不可避免; 成员系统失效将成为常态, 系统稳定性不能依赖对其他成员系统的严苛假设.

(4) 系统边界. 在基于 “还原论” 的软件开发方法中, 人仅仅是系统使用者, 软件仅仅是信息空间的产物. 而在复杂软件系统中, 人和物理世界是构成复杂软件系统的重要因素, 软件、物理世界、社会三者紧密结合.

(5) 可信保证手段. 在基于 “还原论” 的软件开发方法中, 通过软件测试、验证等手段, 在软件上线前解决软件可信性问题. 而在复杂软件系统中, 由于复杂性和开放性, 不可能在开发阶段一劳永逸地解决软件可信性问题.

(6) 系统改进方式. 在基于 “还原论” 的软件开发方法中, 系统的改进是间隔性的, 具有明确的阶段. 而在复杂软件系统中, 系统建设、维护、改进同时持续运作, 软件整体质量在持续改进过程中保持和提升.

传统的软件开发受到大规模工业生产线模式的深刻影响, 旨在以精确和严密的计划, 产出符合事先需求约定的软件产品. 这样产出的软件系统, 从形象上看, 就像封闭的 “烟囱”, 既缺乏结构化的柔性以支持动态的按需重组和再造, 又没有良好的兼容性以实现与其他相关软件 “烟囱” 的互联和互通. 以经典的软件开发方法来指导复杂软件系统的设计和建设, 其结果只能是造就数目众多孤立的、刻板的 “烟囱”, 系统的互操作性、适应性和可演化性都会比较差, 难以形成相互兼容、可持续发展的技术 — 生态环境, 难以实现的长期演化和发展.

近年来, 软件工程领域已经开始逐渐认识到 “还原论” 的局限性, 提出了一些新的软件开发思想和技术, 如面向方面的程序设计 AOP[18]、面向服务体系结构及其软件技术 [19]、面向 Agent 软件工程 [20]、开源软件开发、可信软件工程 [21] 等等. 尤其是最近几年, 如何支持大规模系统的有效开发、灵活部署和持续演化成为人们关注的焦点, 一些新的软件开发方法和技术逐渐成为研究热点, 诸如自适应软件的构造方法 [22]、超大规模系统的软件工程 [1]、软件在线演化使能技术 [23]、面向复杂系统的 “系统联盟” 观点 [24] 等. 本文后续内容将在借鉴和继承这些已有方法思想的基础上, 提出 “成长性构造” 法则和 “适应性演化” 法则.

4 复杂软件系统构造与演化基本法则

复杂软件系统的特点决定了我们无法沿用自顶向下、计划驱动的传统软件开发方法来构建, 本节将给出面向复杂软件系统的 “成长性构造” 和 “适应性演化” 法则基本思路、概念内涵和基本问题.

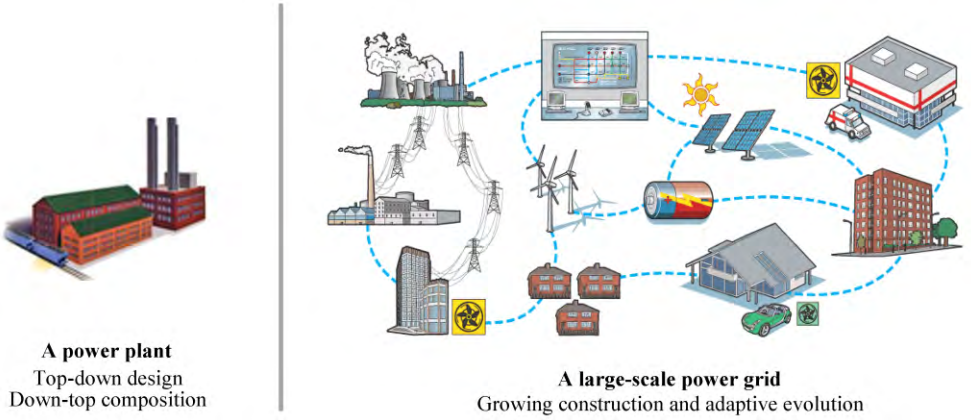

图 3 (网络版彩图) 现实生活中复杂系统的构造

Figure 3 (Color online) Construction of complex systems in real life

4.1 应对挑战的基本思路

经典软件系统和复杂软件系统的区别可以通过能源领域的示例来进行类比 (图 3). 虽然采用自顶向下、计划驱动的方式, 我们可以成功建造一座电站然后交付使用, 但是无法把一个地区或国家级电网所有设施都建造好之后再交付给用户. 大型电网的规模和复杂程度已远远超出了计划驱动模式所能建造的范围, 只有通过持续的 "成长" 和 "演化", 才能逐步形成大型电力网络并维持其良好运转. 大型电网中的个体 (或驱动其建造的人) 都是具有适应能力的自主实体, 他们在法律、规则和制度的引导和约束下进行协同, 推动着电网由小变大、由简陋变成复杂, 不断进化和成长.

不单大型电网这个典型示例, 实际上人类社会、生物圈等其他复杂系统都是在以类似方式构建和演化, 在系统科学领域此类系统被称为 "复杂适应系统"(complex adaptive system)[5], 其基本思想是 "适应性造就复杂性": 系统的成员被看作是具有自身目的、主动积极的个体, 个体能够与环境及其他个体进行交流, 在这种交流过程中 "学习" 或 "积累经验", 据此改变自身的结构和行为方式; 各个底层个体通过交互和交流, 可以在上一层次及整体层次上涌现出新的结构、现象和更复杂的行为, 如新层次的产生、分化和多样性的出现、新的聚集体的形成等. 复杂软件系统需要借鉴上述复杂适应系统已有研究成果, 特别是其形成、成长和演进的模式, 即本文下节将阐述的成长性构造和适应性演化法则.

4.2 成长性构造与适应性演化法则

"成长性构造" 和 "适应性演化" 法则在软件构造层面上强调软件单元的自主性以及单元的新陈代谢和动态连接, 在软件过程层面上强调复杂软件系统为适应环境和需求的变化而实施调整、驱动其在线演化的机制, 从而为复杂软件系统的构造、部署、运行、维护、演化和保障提供有效支撑.

法则 1 (成长性构造法则) 复杂软件系统是在大量自治系统动态连接过程中构造而成的.

成长性构造法则关注的是复杂软件系统的演化使能问题. 该法则强调如下两点: (1) 区别与传统软件开发 "万丈高楼平地起" 的模式, 复杂软件系统的构造是 "成长性" 的. 传统软件开发需要从头生成一个软件系统, 即解决从无到有的问题, 而复杂软件系统是在大量自治系统基础上 "成长" 而来, 这些自治系统中很大一部份是既有的系统, 成长性构造同时兼顾变化的需求以及既有软件系统的现状. (2) "成长" 的主要手段是 "连接", 特别是动态连接. 与人类社会、经济系统等类似, 复杂软件系统的成

员是具有自主性的个体软件单元 (即自治软件系统), 通过这些软件系统的吐故纳新和动态连接来构造出复杂软件系统.

具体到实践层面, 成长性构造法则涉及到如下两个方面的基本问题: (1) 软件模型问题, 即提供什么样的模型来支持复杂软件系统的构造, 进而支持其演化和成长, 软件模型如何体现复杂软件系统构造的生长与代谢特征. (2) 开发方法问题, 即复杂软件系统的构造采用什么样的过程模型, 借鉴什么样的技术来支持系统的构造, 这类系统的采办和管理采用什么样的方法等.

法则 2 (适应性演化法则) 复杂软件系统是在不断适应环境和需求的变化过程中持续演化的.

适应性演化法则关注的是软件演化的性质、动力和过程. 此处, 演化是指对软件系统不断进行更改的活动, 具体表现为软件元素的更替、软件元素的增加、软件元素的删除、软件结构和关系的形成与再造等 [25,26]. 复杂软件系统的复杂性正是来自于其为适应环境和需求变化而持续进行的演化活动, 而其生命力、或者说其能够持续按用户预期提供服务的能力也是来自于适应性演化. 我们强调 "适应性" 演化, 是因为复杂软件系统演化在适应性 [27] 方面有其特殊性, 具体表现为 (1) 传统的软件演化发生在软件系统部署和交付之后的运行阶段, 复杂软件系统的适应性演化通常与系统的构造交织在一起, 覆盖了软件系统的全生命周期. (2) 传统的软件演化大多采用集中的方式周期性或者阶段性的开展, 而复杂软件系统的适应性演化是一个循序渐进、持续和长期的过程. (3) 传统软件系统的演化主要是指软件系统在运行阶段适应需求或者存在问题而实施的改进, 而复杂软件系统的适应性演化是一个基于已有系统、应对各种变化的新陈代谢过程, 它更加强调新旧共存、适应变化、持续成长. (4) 传统软件系统的演化通常采用离线 (off-line) 的方式, 而复杂软件系统的演化需要采用在线 (online) 的方式来进行, 即在保持软件系统持续运行、常态服务的基础上来实现动态适应.

具体到实践层面, 适应性演化法则涉及到如下两个方面基本问题: (1) 理论基础问题, 即如何建立复杂软件系统的适应性演化理论, 复杂软件系统的演化有什么样的规律性, 如何建立复杂软件系统的模型, 描述、分析和验证复杂软件系统适应性演化的规律、机理和特征. (2) 实现机制问题, 这类系统采用什么样的机制来实现长期、持续、不间断、适应性的演化, 使得系统表现出全局性的行为; 复杂软件系统如何与所在的社会系统、物理系统共同实施演化; 具有什么样的生命周期模型等等.

4.3 成长性构造与适应性演化的关系

文献 [28] 认为 "软件演化" 一词可从两个不同视角进行解释: 作为一种技术, 软件演化主要关心对软件进行修改的方法, 也即演化如何实现 (how) 的问题; 作为一种现象, 软件演化主要关心演化性质、谁驱动的演化、演化的效果如何等, 即演化什么 (what) 和为什么演化 (why) 的问题. 将上述两个不同的视角映射到复杂软件系统, 就分别对应成长性构造和适应性演化的概念: 前者关注如何构造软件, 使得软件系统具备演化能力; 后者则关注复杂软件系统中演化活动的内在规律、实施过程等.

图 4 形象地解释了复杂软件的生命周期中成长性构造和适应性演化的内在关系. 软件开发过程可以划分为两个阶段: 初始开发阶段和持续演化阶段. 初始开发阶段完成复杂系统的初始需求分析和体系结构设计, 制订演化的总体规划和监控调节机制, 然后完成初始的工程设计和实现, 进入持续演化的阶段. 这个阶段由循环往复的软件演化任务所组成, 每一个软件演化的步骤都是由环境和需求变化驱动的软件进化 "微循环", 包含环境和需求变化、软件变化规划、成长性构造等阶段. 正是在环境和需求的不断变化, 推动复杂软件系统向着人们希望的方向逐步逼近和进化.

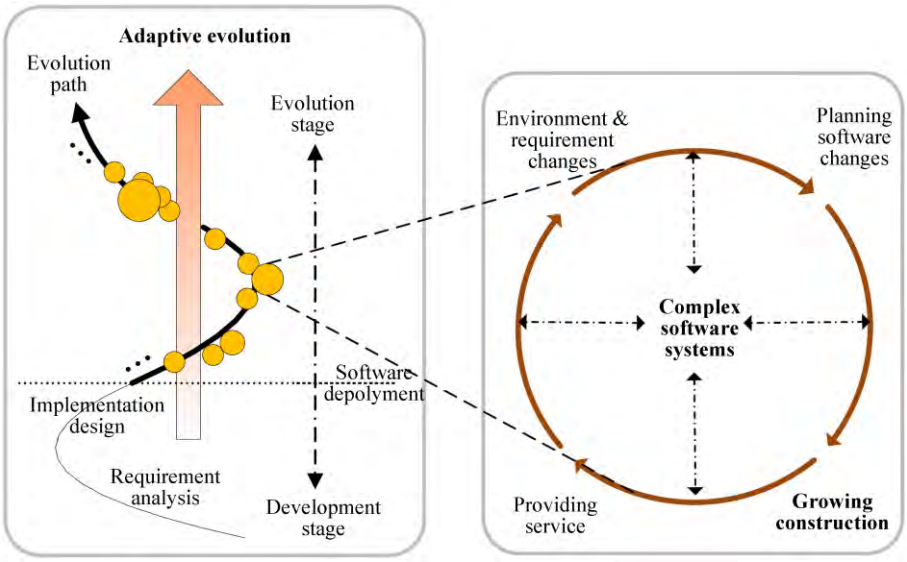

图 4 (网络版彩图) 复杂软件系统成长性构造与适应性演化的关系

Figure 4 (Color online) Relationships between growing construction and adaptive evolution in complex software systems

5 复杂软件系统成长性构造技术体系

本节关注支撑复杂软件系统成长性构造的技术体系. 要通过成长性构造使得软件具备可演化能力, 关键在于: (1) 其软件成员不能仅仅是被动等待调用的功能模块, 而应当能够根据环境和自身状态进行主动调整和适应的自治软件系统 (以下称自主化软件单元, 或软件单元); (2) 软件单元能够在线更新, 而不是部署以后就一成不变, 能够通过新陈代谢来应对环境变化; (3) 软件单元间的连接关系能够动态调整, 从而为群体行为的涌现, 以及系统层面适应性演化的实施奠定基础. 因此, 需要从上述 3 个维度对复杂软件系统成长性构造的软件模型和开发方法提供支持.

5.1 自主化软件单元模型

软件单元是复杂软件系统的基本构造单元. "适应性造就复杂性"[5], 为了支持整个系统的适应性演化, 复杂软件系统中的软件单元需要具备适应环境变化的能力, 这种能力又可以细分为如下 4 个方面的能力:

(1) 环境敏感性 (context-awareness), 即软件单元能够感知外部环境的变化.

(2) 自我敏感性 (self-awareness), 即软件单元能够感知自身状态的变化.

(3) 行为自主性 (autonomy), 即软件单元具有局部于自身的行为控制机制, 可在无需外部指导的情况下实施行为决策.

(4) 变化适应性 (adaptability), 即软件单元能够根据感知到的外部环境或者自身状态, 有针对性地进行调整, 进而展示灵活性.

图 5 给出了复杂软件系统自主化软件单元的参考模型. 在这一参考模型中, 除了封装作为软件单元需要实现的业务功能之外, 增加了实现上述 4 个方面元级能力的功能部件. 一方面, 监控部件提供了对软件单元自身以及外部环境变化的感知能力, 从而为适应和演化提供了依据. 另一方面, "评估 — 分析 — 决策 — 执行" 的回路使得软件单元具备了行为自主性和变化适应性. 以未来能源互联网中的

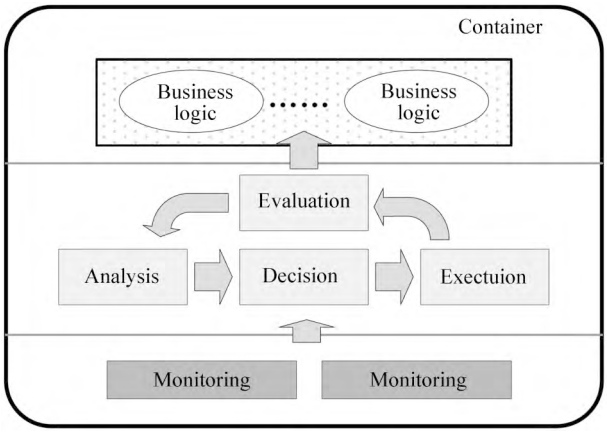

图 5　自主化软件单元参考模型

Figure 5　Reference model for autonomic software units

能源管理自主化软件单元为例, 该单元需要根据感知功能, 了解其计算负载、网络流量等系统状态, 在能耗负载变化时, 动态地评估能量调节的目标和效率, 进而实施本地的决策, 动态加载恰当的逻辑功能部件而加以执行; 针对可能出现的故障和问题, 软件单元应能够自动采取冗余、纠错等可靠性维护策略, 使得系统仍然可以保持相对正常的工作状态. 这些自动决策 — 调整的能力, 使系统展示出良好的灵活性、健壮性和多变性, 以更好地应对不可预知、不确定和不可控的环境变化.

事实上, 图 5 中这种元层 “容器” 对其上基层 “功能” 模块进行调整的思想在经典软件技术中已得到体现. 例如, 操作系统即为应用软件的容器, 操作系统可以根据其所监控到的状态变化 (如内存占用) 调整上层应用软件的运行 (如虚拟内存换页); 云计算环境下的虚拟机管理软件即为虚拟机的容器, 前者可以根据当前环境状态和需求对后者进行调度管理.

软件技术近年的发展则进一步为自主软件单元的构造提供了支撑. 相关使能技术和重要实践包括: (1) 反射 (reflection)[29,30]. 早在 20 世纪 80 年代研究者就已指出, 软件可以建立对自身的描述, 并借助这一描述来操纵自身. 这一概念催生了一系列支持反射的程序设计语言和软件平台. (2) 软件自适应 [22]. 此类软件强调在运行时检测环境变化和自身状态, 据此对行为进行调整, 研究者已经提出了一系列面向自适应软件的体系结构、开发方法和运行基础设施, 典型实例如网构软件中的 “自主构件”. (3) 自主计算 [31]. 自主计算借鉴了自然界中自治系统的思想, 以实现自配置、自优化、自修复、自保护为目标, 其主要成果是基于 MAPE-K 模型的自主元素. (4) 面向 Agent 的软件工程 [32]. Agent 是人工智能领域一个重要分支, 其具有自治性、社交性、主动性等特征, 相关软件开发方法可以为自主化软件单元提供支撑. 随着上述领域的持续发展, 复杂软件系统的成长性构造将具有更为坚实的基础.

5.2　软件单元的在线更新

在自然界, 生物体的适应是通过新陈代谢等机制具体实现的. 映射到复杂软件系统, 则具体表现为软件单元的在线更新等操作 (图 6). 所谓在线更新, 是指在不停止软件服务的前提下, 对构成软件的一个或多个模块进行在线替换的活动 [33], 广义的在线更新还可以包括对软件模块的增删、运行参数和策略的在线配置等操作. 此处强调 “在线”, 是因为复杂软件系统是 “社会 — 技术 — 物理” 交融系统, 在系统层面上往往需要持续提供 7*24 小时服务.

软件的在线更新问题一直是软件工程领域的关注点. 在线更新对复杂软件系统是一个侵入式过

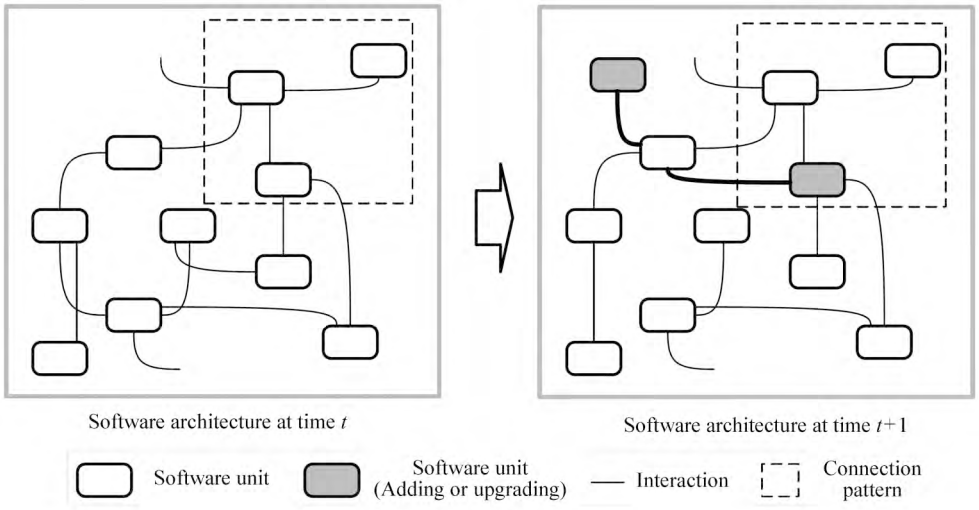

图 6 软件单元的在线更新与动态连接

Figure 6 Online upgrade and dynamic connection of software units

程, 因此首先要解决的问题是更新后系统能否处于正常的状态, 以及软件单元能否从该状态开始继续正常运行. 这一问题被称为一致性问题, 又可以具体分为结构完整性、相互一致性和应用状态不变式等 [34]. 一致性问题牵涉到新系统本身功能逻辑的正确性、新老系统状态过渡的适应性等, 试图完全依靠应用逻辑无关的在线更新方法来确保几乎是不可能的, 但需要成长性构造技术体系在可重用基础设施层面上提供使能机制, 并在方法手段上提供尽可能的一致性保证机制.

5.3 软件单元的动态连接

在社会、经济、生物等系统中, 个体与环境之间、个体与个体之间存在着物质流、能量流和信息流. 这种错综复杂的连接是整体层次上涌现出新结构、现象和更复杂行为的前提和基础. 在复杂软件系统中, 这种连接同样存在 (图 6): 通过单元与其他单元之间的交互和集成, 才能形成规模更大和复杂性更高的系统; 通过单元之间的动态连接, 复杂软件才能有效地适应环境变化.

要实现软件单元的动态连接, 主要需要解决两个方面的问题:

1) 可集成性和互操作性. 在复杂软件系统中, 软件单元存在于不同的软硬件平台、表现出不同的应用行为、采用不同的数据内容和格式、拥有不同的管理策略. 要基于这些软件单元的 “联盟” 来形成复杂软件系统, 首先需要解决的是互操作问题, 而且这种互操作性并不局限于传统互操作协议所关注的消息语法和语义层面, 它在必要时可能涉及到软件行为语义、安全性、管理策略等各个维度.

2) 单元/系统间的动态连接使能. 事实上, 近年来一些被广泛认可的软件工程研究已经在这一方面做出了大量努力. 例如, 在面向服务的计算中, 服务是指对资源进行封装的自治、平台独立的实体, 它们可以被描述、发布、发现和松散绑定. 通过服务的抽象与封装, 快速、廉价、可互操作、可演化和大规模的分布式应用开发成为可能 [35]. 服务发现和服务动态组合实质上就是建立动态连接的一种手段. 更进一步, 在复杂软件系统中, 我们可以从系统架构的层次来看待动态连接问题. 例如, 动态体系结构技术 [36,37] 通过引入运行时软件体系结构模型, 并且将构件和连接子都建模为一阶实体, 通过运行时体系结构的变化来实现软件的新陈代谢和动态连接.

表 1 对前文中提及的、与复杂软件系统成长性构造密切相关的使能技术和已有实践进行了总结.

表 1　复杂软件系统成长性构造相关使能技术

Table 1　Enabling techniques related to growing construction of complex software systems

	Related techniques and practices	Description	Major references
Autonomic software unit	Reflection	Realizing online software self-adjustment by dividing the base layer and meta layer	[29,30]
	Software self-adaptation	Software senses environment and state changes and then adjust itself accordingly	[22]
	Autonomic computing	Realizing software self-configuration, self-optimization, self-healing and self-protection	[31]
	Agent-oriented SE	Constructing software and software systems based on agent-related techniques	[32]
Software unit online upgrade	Software online upgrade	Replacing software components while it is running	[33,34]
Software unit dynamic connection	Component technology	Realizing dynamic connection by adjusting the interaction relationships among components	[12]
	Service-oriented computing	Realizing dynamic connection by service discovery and service dynamic composition	[35]
	Dynamic software architecture	Maintaining a causally-connected online architecture model, and then realizing dynamic connection by modifying connectors in this model	[31]

6　复杂软件系统适应性演化技术体系

成长性构造使得复杂软件系统有可能具备类似于社会、经济等复杂适应系统的结构, 而适应性演化要解决的是如何驱动这种结构运转, 使之能够持续适应环境和用户需求的变化. 这需要从基础理论和实现机制两个基本问题入手.

6.1　适应性演化的基础理论

在适应性演化过程中, 复杂软件系统的行为规律可以从统计确定性和逻辑确定性两个角度寻找. 因此, 统计分析和逻辑演算是适应性演化基础理论的两个重要研究手段, 此外模拟与仿真也有助于我们进一步理解或预测复杂软件系统的行为.

6.1.1　复杂软件系统适应性演化的统计不变性规律

在复杂软件系统中, 所有软件功能单元以及系统整体随着时间的推移, 在包括用户、物理环境的各种复杂因素的作用下, 在原有机能和结构基础上向不同的方向上逐步变异修改, 以适应新的环境. 这种类似生命和社会系统的进化方式, 往往是大规模、复杂和多因素的, 传统机械 “还原论” 方法很难有效刻画, 需要在系统层面上借助软件资源信息挖掘 [38]、复杂网络分析 [39] 等方法, 刻画演化过程中的关联关系, 描绘演化及相关软件工程过程中所遵循的规律.

以软件演化的预测为例, 复杂软件系统的演化需求隐藏在多样化利益相关方使用软件的轨迹中, 需要进行主动的软件演化预测, 以支持软件系统不断演进. 可以结合采用复杂网络分析和数据挖掘的方法, 一方面根据利益相关方使用和构造软件的行为, 以用户偏好和用户标注为主要特征建立用户偏

好模型, 建立软件需求演化预测模式; 另一方面根据以往开发的历史数据, 分析开放社群的程序员的贡献和专长, 为开放社区建立开发者信誉度和社交网络, 从而为建立开发者推荐和激励方法.

另一个示例是复杂软件系统的群体化开发方法. 复杂软件系统需求分析、系统设计和实现、功能和性能测试以及软件软件质量评价等工作, 都不再是单个精英团队或专家所能完成, 而是依靠群体力量, 充分发挥群体智慧 [40,41] 才能达到目标. 群体化开发方法理论和实践将在未来复杂软件系统的构建中扮演重要角色 [42,43], 然而学术界目前对其规律的认识尚处于起步阶段. 应对挑战的途径之一是利用开源软件社区已产生的庞大群体软件开发数据, 对这些数据进行分析, 在实证研究基础上综合运用社会学、认知科学和社会计算的定量科学方法, 认识群体软件开发的动态过程和一般规律, 反过来为复杂软件系统的构造提供指导.

6.1.2 基于逻辑演算的复杂软件系统演化一致性保证技术

软件系统的本质是对问题领域逻辑表达的总括. 在复杂软件系统演化过程中, 每个软件单元的功能定义、模型设计、代码描述、测试用例等等都需要保持逻辑意义上的一致性、完整性、可溯性和正确性. 这就需要逻辑系统和演算方法以形式化的视角, 来严格和精确地定义这些功能的语义变化, 表达其演化的形式序列, 并描绘其极限性质, 实现程序演化过程的逻辑验证和正确维护.

以开放逻辑 [44] 为例, 它通过引入修正演算, 根据事实的反驳, 修正原有的形式理论, 并使其具有可达性、可靠性和完全性. 复杂软件演化的软件版本更新理论和一致性维护方法可以以开放逻辑作为研究手段. 修正演算可以描绘各个软件单元的变化过程, 定义其适应的目标和修改的约束条件, 在与开发人员交互过程中, 帮助开发人员分析和判断可行的程序修改路径和重组方式, 从而完成约束条件下的程序模块的增删和修改. 同时修正演算还将记录每个软件单元的修改版本和测试轨迹, 从而形成完整的修正演化历史, 对演化的趋势和极限性质, 给出形式化的定义.

6.1.3 面向复杂软件系统演化的群体行为模拟和仿真方法

为了更好地分析复杂软件系统演化规律, 可以交叉和借鉴社会组织学的思想, 将复杂软件系统的体系结构抽象为组织结构, 将体系结构模式抽象为组织模式, 将体系结构的约束抽象为组织法规, 借鉴准则调控的多 Agent 系统 [45] 和虚拟组织 [46] 等领域的研究成果, 为复杂软件系统内部和外部的各类动态连接提供方法指导和必要使能机制.

模拟和仿真是认识客观事物规律的重要手段, 可以构造基于 Agent 的群体程序进化仿真和模拟环境, 实现相关可视化展示工具, 模拟各种调控机制下的软件进化行为, 探索其内在规律. 仿真环境将可以对程序系统、参与程序研制的人员和开发环境等主要的对象进行建模, 实现对软件开发过程的细粒度仿真, 检验不同的需求定义、软件设计方式、软件开发组织模式以及过程管理的策略等对复杂软件系统开发的周期、效率、成本和风险等的影响. 通过仿真环境, 复杂软件系统的架构师和设计师可以预先模拟各种不同组合情况, 确定最佳的项目实施方案, 对成本、风险和开发进度等有较为准确的预测.

6.2 适应性演化实现机制

适应性演化实现机制由软件运行时监控、海量监控数据的汇聚分析、多尺度行为决策、快速恢复和在线调整等环节组成 (如图 7 所示), 需要在软件单元、系统乃至于系统的系统等不同的尺度上形成 “监控 — 分析 — 决策 — 调整” 的适应回路, 建立人机协同驱动软件适应活动的机制. 其中, 在线调整和演化实施的具体实现主要表现为软件单元内部组成模块的在线更新、软件单元间连接关系的动态调整等, 可以参见本文第 5 节已有描述. 本节主要关注监控使能、态势评估和演化决策 3 个环节.

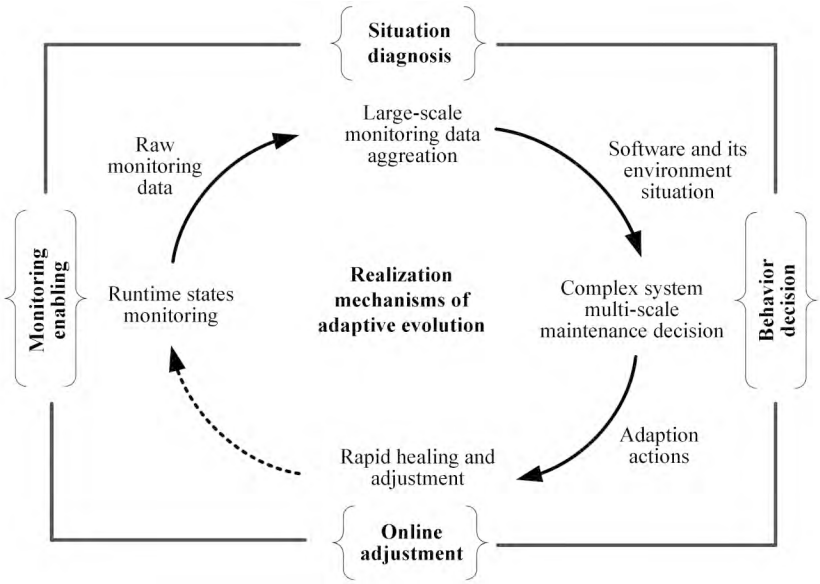

图 7　复杂软件系统的适应性演化实现机制

Figure 7　Realization mechanisms of complex software system adaptive evolution

6.2.1　复杂软件系统监控使能技术

所谓监控使能, 是指通过一组方法和技术, 使得复杂软件系统在系统层面上具备环境和状态监控能力. 这一环节涉及到几个关键问题: 如何表达复杂软件系统的监控数据, 如何在软件中实现监控逻辑与业务逻辑的集成, 如何实现监控数据到监控基础设施的无缝接入.

在监控使能环节, 常见的运行状态和环境数据建模方法包括一阶逻辑、本体、图等 [47]. 复杂软件系统中的软件要素和要素间交互关系要能够被监控, 需要从监控软件开发方法和监控软件动态集成两个方面开展工作: 前者使得软件要素具备监控能力 [48], 可以通过面向方面编程 (AOP)、封装、插桩等方法实施监控代码注入; 后者使得软件要素可以动态接入到监控基础设施中, 从而支持复杂软件系统在监控维度上的持续成长和演化. 在此基础上, 还可以通过主元素分析 (PCA)[49] 等方法对所收集到监控数据进行降维, 为后续态势评估奠定基础.

6.2.2　复杂软件系统态势评估技术

在复杂软件系统的态势评估环节, 将对监控数据进行汇聚和分析, 实现 “去粗取精、去伪存真”, 进而对软件的高层运行状态进行评估, 对各类异常进行检测和诊断, 为适应性演化的决策环节提供依据. 由于复杂软件系统规模巨大, 在监控过程中会实时产生以流的形式所提供的大量环境和状态数据, 这些数据之间存在着千丝万缕的联系, 对其处理需要大数据分析、流数据处理等相关技术的介入. 例如, 复杂软件系统中的 7∗24 小时持续服务的后端往往以多构件、多实例的方式部署, 通过多个实例监控数据的关联分析, 有可能快速检测到异常, 包括设计阶段未知的异常 [50,51]; 当某种异常发生过以后, 可以建立其异常 “签名” 或 “指纹”, 后续就可以通过特征提取和匹配来检测相应异常是否发生 [52]; 在数据量足够大的前提下, 可以通过结合贝叶斯模型和决策树算法预测软件失效并进行前瞻性管理 [53]; 等等. 此外, 实时监控数据的分析还可以与系统日志数据的处理相结合. 系统日志包括了请求踪迹、告警、资源利用记录等, 可以通过机器学习、统计分析等方式对实时监控数据、日志等在不同尺度上进

表 2 复杂软件系统适应性演化构造相关使能技术

Table 2 Enabling techniques related to adaptive evolution of complex software systems

	Related techniques and practices	Description	Major references
Fundamental theory of adaptive evolution development	Mining software repository	Mining software repository and finding its objective laws	[38]
	Collective software intelligence and in a crowdsourcing manner	Driving software development by collective	[40∼43]
	Open logic and other formal methods	Modeling and analyzing software evolution process by formal methods	[44]
	Multi-agent theory and its simulation	Constructing simulation environments for large-scale distributed systems based on the multi-agent theory	[45,46]
	Complex network theory based on agent-related techniques	Constructing software and software systems	[39]
	Complex adaptive system theory	Applying the complex system theory into software development	[5]
Realization mechanism of adaptive evolution	Monitoring data modeling	Modeling monitoring data by ontology, graph and other means	[47]
	Monitoring prober instrumentation	Integrating monitoring probers with existing and newly-developed software	[48]
	Software anomaly diagnosis	Localizing the root cause of software anomalies	[50∼53]
	Human-machine cooperated adaptation	Driving software adaptation in a human-machine cooperated manner	[54,55]
	Policy-based adaptation	Driving software adaptation by predefined policies (such as if-then rules or utility functions)	[27]
	Control and decision theory	Applying control and decision theory into software adaptation decision	[27]

行汇聚处理, 实现异常识别、隐患和故障定位.

6.2.3 复杂软件系统适应性演化的决策技术

如何根据环境和状态变化, 做出软件在线调整和演化的恰当决策, 一直是软件适应和演化领域的研究热点. 传统的研究一般都以追求完全的 “自动化” 为目标, 但在复杂软件系统中, 首先运行环境和系统状态都可能超出开发阶段的预期, 要完全依赖软件自身作出决策显然不现实 [54,55]; 其次, 如前所述, 人本来就是复杂软件系统的有机组成成分, 会在系统运行和演化过程中发挥重要作用. 因此, 复杂软件系统适应性演化应当是软件自动决策和人工决策相结合的方式, “人在系统中” 是其重要的特点.

在适应性演化的自动决策技术方面, 可以采用规则策略、效用计算等方式, 以及人工智能中动态规划、模糊逻辑、遗传算法等相关技术. 此外, 控制理论、决策论、仿生学、机制设计等交叉学科的成果也是重要的借鉴对象 [27]. 在适应性演化的人工决策方面, 主要是要为 “人在系统中” 提供必要的软件工程使能机制和方法, 例如人对监控结果的观察手段和必要时对系统进行调整的工具支持 [54]. 此

外, 复杂软件系统的适应性演化往往发生在软件单元、系统等多个尺度上, 需要考虑如何在多个软件单元组成的群体层面上形成恰当的适应性演化决策, 以及不同层次上演化决策如何相互影响等.

表 2 给出了与复杂软件系统适应性演化相关的主要使能技术和已有实践.

7 结论

复杂软件系统是 “系统之系统”、“信息 — 物理” 融合系统和 “社会 — 技术” 交融系统. 传统基于 “还原论” 思想的方法已经无法有效支撑此类软件系统的构造和演化. 本文针对这一挑战, 提出复杂软件系统 “成长性构造” 和 “适应性演化” 法则, 在软件构造层面上强调软件单元的自主性以及单元的新陈代谢和动态连接, 在软件过程层面上强调系统为适应环境和需求的变化而实施调整、不断在线演化. 对于软件技术而言, “可运行” 才能 “降生”, “可应用” 才算 “成熟”. 本文仅仅给出复杂软件系统成长性构造与适应性演化的基本框架, 要完全实现这一目标, 尚有诸多理论和技术挑战尚有待我们去克服.

参考文献

1 Northrop L, Feiler P, Gabriel R, et al. Ultra-Large-Scale Systems: The Software Challenge of The Future. Carnegie Mellon University Technical Report. 2006

2 Boehm B. A view of 20th and 21st century software engineering. In: Proceedings of International Conference on Software Engineering, Shanghai, 2006. 12–29

3 Cliff D, Northrop L. The global financial markets: an ultra-large-scale systems perspective. In: Proceedings of Workshop of Development, Operation, and Management of Large-Scale Complex IT Systems, Monterey, 2012. 29–70

4 Huang A Q, Crow M L, Heydt G T, et al. The future renewable electric energy delivery and management (FREEDM) system: the energy internet. P IEEE, 2011, 99: 133–148

5 Holland J H. Hidden Order: How Adaptation Builds Complexity. New York: Perseus Books, 1995

6 Lee E A. Cyber-physical systems-are computing foundations adequate. In: Proceedings of NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap, Austin, 2006

7 Maier M W. Architecting principles for systems-of-systems. Syst Eng, 1998, 1: 267–284

8 Qian X S, Yu J Y, Dai R W. A new discipline of science—the study of open complex giant system and its methodology. Chin J Nature, 1990, 13: 3–10 [钱学森, 于景元, 戴汝为. 一个科学新领域 —— 开放的复杂巨系统及其方法论. 自然杂志, 1990, 13: 3–10]

9 Naur P, Randell B. Software Engineering: Report of a Conference Sponsored by the NATO Science Committee. NATO Technical Report. 1969

10 Zelkowitz M V. Perspectives in software engineering. ACM Comput Surv, 1978, 10: 197–216

11 Meyer B. Object-oriented software construction. New York: Prentice hall, 1988

12 Syperski C. Component Software: Beyond Object-Oriented Programming. Boston: Addison-Wesley, 2002

13 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Englewood Cliffs: Prentice Hall, 1996

14 Bernstein P A. Middleware: a model for distributed system services. Commun ACM, 1996, 39: 86–98

15 Fuggetta A. Software process: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, Limerick, 2000. 25–34

16 Royce W W. Managing the development of large software systems. In: Proceedings of IEEE Western Electronic Show and Convention, Los Angeles, 1970

17 Sommerville I. Software process models. ACM Comput Surv, 1996, 28: 269–271

18 Kiczales G, Lamping J, Mendhekar A, et al. Aspect-Oriented Programming. Berlin: Springer, 1997

19 Tsai W. Service-oriented system engineering: a new paradigm. In: Proceedings of IEEE International Workshop on Service-Oriented System Engineering, Beijing, 2005. 3–6

20 Wooldridge M. Agent-based software engineering. IEE P Softw, 1997, 144: 26–37

21 Devanbu P T, Fong P W, Stubblebine S G. Techniques for trusted software engineering. In: Proceedings of International Conference on Software Engineering, Kyoto, 1998. 126–135

22 Cheng B H, de Lemos R, Giese H, et al. Software Engineering for Self-Adaptive Systems: a Research Roadmap. Berlin: Springer, 2009. 1–26

23 Lehman M M. Laws of software evolution revisited. In: Proceedings of European Workshop on Software Process Technology, Nancy, 1996. 108–124

24 Sommerville I, Cliff D, Calinescu R, et al. Large-scale complex IT systems. Commun ACM, 2012, 55: 71–77

25 Wang H M, Shi P C, Ding B, et al. Online evolution of software services. J Comput, 2011, 34: 318–328 [王怀民, 史佩昌, 丁博, 等. 软件服务的在线演化. 计算机学报, 2011, 34: 318–328]

26 Lehman M M. Programs, life cycles, and laws of software evolution. P IEEE, 1980, 68: 1060–1076

27 Ding B, Shi D X. Software Adaptability Techniques: From Individual Adaptation to Collective Adaptation. Beijing: Science Press, 2013 [丁博, 史殿习. 软件适应性技术: 从个体适应到群体适应. 北京: 科学出版社, 2013]

28 Lehman M M, Ramil J F, Kahen G. Evolution as a noun and evolution as a verb. In: Proceedings of Workshop on Software and Organisation Co-evolution, London, 2000

29 Smith B C. Reflections and semantics in a procedural language. Dissertation for Ph.D. Degree. Cambridge, MA: Massachusetts Institute of Technology, 1982

30 Maes P. Concepts and experiments in computational reflection. ACM Sigplan Notices, 1987, 22: 147–155

31 Horn P. Autonomic Computing: IBM's Perspective on the State of Information Technology. IBM Technical Report, 2001

32 Wooldridge M, Ciancarini P. Agent-oriented software engineering: the state of the art. In: Proceedings of International Workshop on Agent-Oriented Software Engineering, Montreal, 2001. 1–28

33 Toby B, Mark D. Reconfiguration and module replacement in argus: theory and practice. Software Eng J, 1993, 8: 102–108

34 Almeida J P A, Wegdam M, Van Sinderen M, et al. Transparent dynamic reconfiguration for CORBA. In: Proceedings of International Symposium on Distributed Objects and Applications, Rome, 2001. 197–207

35 Papazoglou M P, Traverso P, Dustdar S, et al. Service-oriented computing: state of the art and research challenge. IEEE Comput, 2007, 40: 38–45

36 Mei H, Huang G, Zhao H, et al. A software architecture centric engineering approach for internetware. Sci China Ser-F Inf Sci, 2006, 49: 702–730

37 Bradbury J S, Cordy J R, Dingel J, et al. A survey of self-management in dynamic software architecture specifications. In: Proceedings of ACM SIGSOFT Workshop on Self-managed Systems, Newport Beach, 2004. 28–33

38 Kagdi H, Collard M L, Maletic J I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J Softw Maint Evol R, 2007, 19: 77–131

39 Wang X F, Li X, Chen G R. Complex Networks Theory and its Application. Beijing: Tsinghua University Press, 2006 [汪小帆, 李翔, 陈关荣. 复杂网络理论及其应用. 北京: 清华大学出版社有限公司, 2006]

40 Lévy P, Bonomo R. Collective Intelligence: Mankind's Emerging World in Cyberspace. New York: Perseus Publishing, 1999

41 Mistrík I, Grundy J, Van der Hoek A, et al. Collaborative Software Engineering: Challenges. Berlin: Springer, 2010

42 Wang H M, Yin G, Xie B, et al. Research on network-based large-scale collaborative development and evolution of trustworthy software. Sci China Inf Sci, 2014, 44: 1–19 [王怀民, 尹刚, 谢冰, 等. 基于网络的可信软件大规模协同开发与演化. 中国科学: 信息科学, 2014, 44: 1–19]

43 Huhns M N, Li W, Tsai W T. Cloud-Based Software Crowdsourcing. Dagstuhl Seminar Technical Report 13362, 2013

44 Li W. An open logic system. Sci China Ser A, 1993, 36: 362–375

45 Boella G, Van Der Torre L, Verhagen H. Introduction to normative multiagent systems. Comput Math Organ Theory, 2006, 12: 71–79

46 Mowshowitz A. Virtual organization. Commun ACM, 1997, 40: 30–37

47 Strang T, Linnhoff C. A context modeling survey. In: Proceedings of Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, 2004

48 Liu D H, Guo C G, Wang H M, et al. Monitoring enabled distributed software construction method. J Software, 2011, 22: 2610–2624 [刘东红, 郭长国, 王怀民, 等. 监控使能的分布式软件系统构造方法. 软件学报, 2011, 22: 2610–2624]
49 Jolliffe I T. Principal Component Analysis. Berlin: Springer, 1986
50 Sharma B, Jayachandran P, Verma A, et al. CloudPD: problem determination and diagnosis in shared dynamic clouds. In: Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks, Budapest, 2013. 1–12
51 Kang H, Chen H, Jiang G. PeerWatch: a fault detection and diagnosis tool for virtualized consolidation systems. In: Proceedings of International Conference on Autonomic Computing, Washington, 2010. 119–128
52 Bodik P, Goldszmidt M, Fox A, et al. Fingerprinting the datacenter: automated classification of performance crises. In: Proceedings of European Conference on Computer Systems, Paris, 2010. 111–124
53 Guan Q, Zhang Z, Fu S. Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems. J Commun, 2012, 7: 52–61
54 Ding B, Wang H M, Shi D X, et al. Taming software adaptability with architecture-centric framework. In: Proceedings of IEEE International Conference on Pervasive Computing and Communications, 2010. 145–151
55 Ding B, Wang H M, Shi D X. Constructing software with self-adaptability. J Software, 2013, 24: 1981–2000 [丁博, 王怀民, 史殿习. 构造具备自适应能力的软件. 软件学报, 2013, 24: 1981–2000]

Growing construction and adaptive evolution of complex software system

WANG HuaiMin[1], WU WenJun[2], MAO XinJun[1], DING Bo[1*], GUO ChangGuo[3] & LI Wei[2]

1 *College of Computer, National University of Defense Technology, Changsha* 410073*, China;*
2 *College of Computer, Beijing University of Aeronautics and Astronautics, Beijing* 100191*, China;*
3 *Chinese Electronic Equipment System Cooperation, Beijing* 100039*, China*
*E-mail: dingbo@nudt.edu.cn

Abstract With the pervasive development of information network technology, complex software system is emerging as a new ubiquitous software paradigm. This kind of software systems often consist of a large amount of autonomic software systems via coupling and interconnection. They exhibit the common characteristics of system of systems, cyber-physical systems and socio-technical systems, including member heterogeneity, open boundary, behavior emergence and constant evolution. These properties invalidate the fundamental assumption of conventional software engineering methodology based on reductionism, thus rendering it not suitable for the construction of complex software systems. In order to effectively support construction, deployment, running and maintenance of complex software systems, this paper studies the connotation, characteristics and basic properties of complex software systems and discusses the challenges about the construction and evolution of complex software systems. By observing the common patterns in current complex systems such as the Internet, life systems, social and economy systems, we propose two fundamental principles of complex software system development, namely the law of "growing construction" and the law of "adaptive evolution". On the basis of extensive elaboration on the major research problems and supporting technologies of those two laws, we further introduce new software methodology and architecture for construction and development of complex software systems.

Keywords complex software system, growing construction, adaptive evolution, software development, software evolution

WANG HuaiMin was born in 1962. He received the Ph.D. degree in computer science from the National University of Defense Technology, Changsha, in 1992. Currently, he is a professor at National University of Defense Technology. His research interest includes distributed computing, internet computing, software evolution and software engineering. He is a CCF Fellow and a member of IEEE and ACM.

MAO XinJun was born in 1970. He received the Ph.D. degree in software engineering from the National University of Defense Technology, ChangSha, in 1998. Currently, he is a professor at the National University of Defense Technology. His research interests include software engineering, multi-agent system, self-adaptive and self-organized system, and autonomic computing. He is a member of IEEE, ACM and CCF.

WU WenJun was born in 1973. He received the Ph.D. degree in computer science from the Beihang University, Beijing, in 2001. Currently, he is a professor at the Beihang University. His research interests include crowdsourcing, green computing, cloud computing, eScience and cyberinfrastructure, and multimedia collaboration. He is a member of CCF.

DING Bo was born in 1978. He received the Ph.D. degree in computer science from the National University of Defense Technology, ChangSha, in 2010. Currently, he is a lecturer at the National University of Defense Technology. His research interests include distributed computing and adaptive software. He is a member of IEEE, ACM and CCF.