软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(1):62−79 [doi: 10.13328/j.cnki.jos.005645]

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

软件过程与管理方法综述*

荣国平[1], 张 贺[1], 邵 栋[1], 王 青[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 210093)

[2](中国科学院 软件研究所,北京 100190)

通讯作者: 邵栋, E-mail: dongshao@nju.edu.cn

摘 要: 工程化软件开发需要对软件开发整个过程进行有效的组织和管理,由此产生了一系列软件开发组织和管理方法,其主要目的是形成一种载体,用以积累和传递关于软件开发的经验教训.然而,由于软件开发的一些天然特性(比如复杂性和不可见性)的存在,使得描述软件开发过程的软件开发与组织方法也天然地带着一定的抽象性.由此带来了很多概念上的误导和实践中的争论,影响了上述目的的达成.例如,对于究竟该如何选择和定义合适的软件开发过程以更好地满足某个特定项目的要求,目前仍然缺少可靠的手段.甚至有些面向工业界的调研报告表明:在实际软件项目开发中,过程改进(例如引入新的工具或者方法)的主要驱动力是佚闻.试图厘清软件组织与管理话题的若干核心概念,系统梳理软件组织和管理方法的特征,并且以软件发展的历史为主线,介绍软件组织与管理方法的历史沿革,整理出这种历史沿革背后的缘由.在此基础上,讨论和总结若干发现,以期为研究者和实践者提供参考.

关键词: 软件工程;软件过程;软件项目管理;软件过程管理

中图法分类号: TP311

中文引用格式: 荣国平,张贺,邵栋,王青.软件过程与管理方法综述.软件学报,2019,30(1):62−79. http://www.jos.org.cn/1000-9825/5645.htm

英文引用格式: Rong GP, Zhang H, Shao D, Wang Q. Survey of process and management approaches for software development. Ruan Jian Xue Bao/Journal of Software, 2019,30(1):62−79 (in Chinese). http://www.jos.org.cn/1000-9825/5645.htm

Survey of Process and Management Approaches for Software Development

RONG Guo-Ping[1], ZHANG He[1], SHAO Dong[1], WANG Qing[2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Insititute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Software development via engineering approaches requires effective organization and management of the entire software development process, which resulted in a series of methods to organize and manage the development of software systems. Its original intention is to form a carrier to accumulate and deliver experiences and lessons learned about software development. However, due to some of the intrinsic characteristics (e.g. complexity and invisibility) of software, the software development and organization methods that describe the software development process also naturally have a certain degree of abstraction. As a consequence, many conceptual misleadings and meaningless debates in practice have affected the achievement of the above objectives. For example, it is still lack of reliable means for how to select and define the right software development process to better meet the requirements of a particular project. Moreover, some industry-oriented research reports indicate that the main driving force for process improvement (i.e., the introduction of new tools or methods) in real software projects is anecdotes. This study attempts to clarify some core concepts related to software

* 基金项目: 国家自然科学基金(61572251); 南京大学计算机软件新技术国家重点实验室开放课题(KFKT2017A13)

Foundation item: National Natural Science Foundation of China (61572251); State Key Laboratory for Novel Software Technology (Nanjing University) (KFKT2017A13)

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: 2018-07-02; 修改时间: 2018-08-15, 2018-09-25; 采用时间: 2018-10-08; jos 在线出版时间: 2018-11-22

CNKI 网络优先出版: 2018-11-23 07:17:58, http://kns.cnki.net/kcms/detail/11.2560.TP.20181123.0717.001.html

organization and management, then systematically sorts out the characteristics of software organization and management methods. Meanwhile, based on the history of software development, the reason behind of the method evolution is figured out. On this basis, several findings are discussed and summarized in order to provide references for researchers and practitioners.

Key words: software engineering; software process; software project management; software process management

1 背景与概念介绍

一个完整的计算机系统通常是由硬件以及运行其上的软件所组成.这其中,软件从诞生开始,其规模以及在一个完整计算机系统中所占的比重一直呈现上升趋势.类似硬件产品的“摩尔定律”(即:硬件产品的性能,每隔约 18 个月性能翻番,成本下降),软件产业也有一个类似的“摩尔定律”,即:类似功能的软件产品的规模每隔 18 个月,其规模(比如代码行)会翻倍,而用户获取该软件或者服务的代价将会下降[1].Humphrey 收集整理了一些典型软件产品中规模(单位为千行代码)随着时间推移的变化趋势(如图 1 所示)[2].我们从中可以发现:在一些系统软件产品(例如,IBM 的软件产品和微软 NT 产品)和应用软件产品(例如,航空领域软件系统和电视机中的嵌入式软件系统)中,软件系统的规模都随着时间的推移呈现出明显的上升趋势.“摩尔定律”给软件系统的开发和维护带来很多负面影响:首先,软件规模的不断扩大,会使得软件开发益发困难,由此带来成本超支、交付延后以及质量低劣等问题;其次,为了缓解摩尔定律的影响,尤其为了尽可能地避免收益下降,软件供应商还需尽快交付产品或者服务,这将使得前述由于规模扩大所带来的影响更加突出.此外,如果考虑到软件在一个计算机系统中的比重也在不断增加,那么软件行业的“摩尔定律”对整个计算机系统的研发和交付的影响也将逐渐增加,进而对由“软件定义的世界”的社会生活各个方面带来巨大影响.

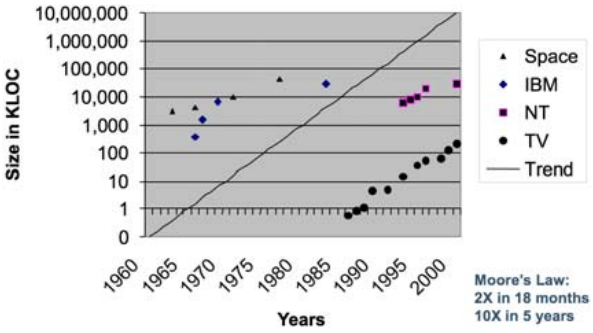

Fig.1 Moore’s law in software industry[2]

图 1 软件行业的摩尔定律[2]

从一定程度上看,上述各个问题就构成了所谓的“软件危机”的关键要素.为了解决上述问题,对软件开发进行有效的组织与管理非常重要,由此诞生并且演化了一系列的所谓软件过程与方法.这其中,诸如瀑布模型、敏捷方法、CMMI 模型以及 DevOps 等也已经成为软件行业几乎无人不知的概念.然而,如此众多的模型和方法的提出,能否解决软件开发(以及运维)的组织与管理问题?或者在多大程度上解决了这些问题?这就需要我们深入剖析上述模型或者方法的特征,系统分析和理解这些模型和方法被提出的缘由,并在此基础上进行合理的决策、规划和组织.本文正是基于这个目的,试图在软件工程概念提出 50 周年之际,将软件开发的组织与管理相关方法的脉络梳理清楚,从而在一定程度上回答上文提出的问题.

要将此事阐述清楚,我们首先需要对两个非常容易引起混淆的概念进行明确的定义和解释,即软件项目管理和软件过程管理(改进).从一定程度上说,目前在软件开发的组织与管理中出现的大量方法上的争议和由此带来的各种误解以及混乱,其根源就在于对上述两个概念区分不够清晰.

1.1 软件项目管理

软件项目管理被称为规划和带领项目团队的艺术和科学[3].顾名思义,其管理的对象是各类软件项目.具体而言,软件项目管理是应用方法、工具、技术以及人员能力来完成软件项目,实现项目目标的过程.整个软件开发过程中的目标识别、状态跟踪以及偏差纠正是软件项目管理的三大核心要素.完整的软件项目管理包含很多内容,例如成本和工作量的估算、计划和进度跟踪调整以及风险分析与控制等,都属于软件项目管理的范畴(可以参考第 6 版 PMBOK 来了解项目管理知识领域).尽管抽象概念上具有相似性(例如,所有管理活动都应该包含目标识别、状态跟踪以及偏差纠正这三大核心要素),软件项目管理的具体方法和实践会因为项目特征的差异呈现出不同的特点.软件项目管理需要借鉴一些本领域或者其他领域的经验教训,由此产生了一些用来描述这些经验和教训的概念,例如软件过程、生命周期模型等.下面分别加以介绍.

- 软件过程

软件过程是为了实现一个或者多个事先定义的目标而建立起来的一组实践的集合[4].这组实践之间往往有一定的先后顺序,作为一个整体来实现事先定义的一个或者多个目标.需要特别指出的是:这里面所提及的一个或者多个目标就是软件项目管理试图要实现的目标,例如成本、工期以及质量等典型目标.因此,好的软件项目管理离不开合理的软件过程.一般来说,当我们讨论过程改进的相关概念时,技术(包括工具)革新、人员培训以及流程优化等均属于软件过程改进的内容.因此,我们认为软件过程也应该包括上述 3 个方面在内.为了以示区别,我们可以将软件过程区分为狭义的过程和广义的过程:狭义的过程就是前文描述的一组有先后顺序的实践;广义的软件过程则应该包括技术、人员以及狭义过程这 3 部分,如图 2 所示.过程的作用则不仅仅是这个三角形中的一极,它更加关键的作用是连接技术和人员的粘合剂.只有将技术、人员以及狭义过程三者融合成一个整体,才是一个真正可以指导工作的软件过程.在某些文献中,会使用软件开发方法[5,6]或者软件开发过程等术语,其含义类似于狭义的软件过程.从这个意义上说,诸如净室 Cleanroom 方法[7]、极限编程 XP(extreme programming)[8]方法、SCRUM[9]方法、DSDM(dynamic systems development method)方法[10]、FDD(feature driven development)方法[11]、Gate 方法[12]等均为软件(开发)过程;而更一般地,敏捷软件过程/方法、轻量型过程/方法以及重型过程/方法等描述也是恰当的(使用这些形容词来描述软件过程的特征是否合理,我们在下文还将进一步讨论).一个软件过程既可以覆盖从需求到交付的完整生命周期,也可以仅仅包括某些特定开发阶段.例如,一个实现(implementation)过程就可能包括详细设计、编码、代码评审、编译以及单元测试等更为细小的开发步骤.甚至,为了确保代码评审工作的完成质量,我们也可以进一步定义、管理和改进代码评审过程.

Fig.2 A generalized software process includes techniques, people and narrowly defined software processes

图 2 广义软件过程包括技术、人员以及狭义过程

- 生命周期模型

生命周期模型与软件开发过程(即狭义软件过程)的概念很像.在很多场合,甚至也会直接以某个特定的生命周期模型来指代某种软件开发过程或方法.例如最为常见的瀑布开发方法,既是指一种以单一线性顺序为特征的软件过程,同时也是指一种常见的生命周期模型.但是,这两者之间还是有一些细小的区别.

- 首先,生命周期模型是对一个软件开发过程的人为划分.同样的软件开发过程,因为目的不一样,可能会被划分和命名成不同的生命周期模型.例如,为了突出后续验证确认阶段与上游开发阶段之间的对应关系,一个单一顺序的软件开发过程可以划分和命名成 V 模型.但是,同样的软件开发过程,通常也可以划分成为一个典型的瀑布模型;

- 其次,生命周期模型是软件开发过程的主框架,是对软件开发过程的一种粗粒度划分.所有的生命周期模型往往只描述一个完整软件开发过程的主要阶段,而不会给出某个阶段的内部具体细节.比如:在一个典型的瀑布模型中,会包含一个需求分析阶段;然而需求分析阶段的具体步骤和流程,则一般不会在生命周期模型中给出描述;
- 最后,生命周期模型往往不包括技术实践.生命周期模型的提出,主要是为了使大规模的软件开发变得易于控制和管理,本质上仍然是分治法策略的应用.因此,生命周期模型中往往只定义管理实践(例如,项目计划、风险分析等);而一些典型的技术类实践(例如,重构、测试驱动开发等)则不会出现在生命周期模型的描述当中.

典型的生命周期模型包括瀑布模型、迭代式模型、增量模型、螺旋模型以及原型法等.这些生命周期模型可以视作一个经典软件过程的模版或者主框架,为进一步定义可行性更高、更具指导意义的软件过程提供了基础.当然,生命周期模型的这种特点,也使得当我们试图理解和研究某个软件项目开发的组织与管理时,仅仅考察所使用的生命周期模型是远远不够的,还应深入到具体的软件开发过程和开发实践中.

1.2 软件过程管理

我们在很多文献中都能看到项目管理应当包括对软件过程的管理这样的论述.事实上,这种说法并不恰当.顾名思义,软件过程管理,其管理对象正是上一节讨论的软件过程,而这种管理的直接目的是为了让软件过程在开发效率、质量等方面有着更好的性能绩效(performance).如果将软件项目管理视作传统行业的产品生产管理的话,软件过程管理则应该是对生产该产品的流水线的设计、建设、维护、优化以及升级改造.软件过程管理一般包括了软件过程的建立、执行、监控、评估以及改进等活动.为了更好地开展软件过程管理,我们同样需要积累相关活动的经验教训,形成了若干可以参考的模型和方法,这其中最著名的软件过程管理参考模型之一可能就是能力成熟度模型 CMM[13]以及其后续的集成模型 CMMI[4,14].探讨一下该模型在实践中被误用和曲解的若干场景,可能有助于我们进一步理解软件过程管理的概念.

- 首先,CMM/CMMI 并不是一种具体的软件过程或者软件开发方法.

在不少文献中,CMM/CMMI 都被视作一种官僚化和教条主义的重型软件过程,并且与当前软件开发大环境格格不入.事实上,按照 CMM/CMMI 模型的要求,一个软件组织应当定义适应本软件组织特点的软件过程,并且不断优化该过程,以更好地实现软件组织的商业目标.然而实践中,软件组织往往为了迎合基于 CMM/CMMI 模型的“证据验证(verification)”(另外一种评估方法称为“证据发现(discovery)”,因为实施代价高昂,极少被采用)评估方法,刻意准备大量文档化证据,导致 CMM/CMMI 被视作在软件项目管理中必须满足的某种标准(这一点类似于几乎所有的 ISO 系列标准的贯标审核).显然,这是对 CMM/CMMI 模型意图和使用方法的曲解.

- 其次,CMM/CMMI 并不能作为检验软件过程优劣的标准.

实践中,很多人会将达到一定成熟度水平视作某个软件组织的研发能力,并且试图进行横向比较,认为成熟度较高的企业,其研发能力应该强于成熟度较低的企业.在 Leon J. Osterweil 那篇影响极大的论文中,也将 CMM/CMMI 视作是检验过程优劣的标准[15].而事实上,由于企业所处的环境以及要实现的目标等方面的差异,过程改进对于不同企业的含义是不一样的.因此,成熟度等级不适宜脱离企业环境直接横向比较;同处于相同的成熟度等级,也并不能说明这些企业的研发能力也是相同的.

- 最后,将 CMM/CMMI 与其他软件过程或者软件开发方法的比较是没有任何意义的.

很多人习惯于将 CMM/CMMI 作为敏捷方法的对立面[16−18],试图来解释和说明敏捷方法的优势.事实上,这种语境之下所谓的“CMM/CMMI 方法”其实已经不是一个过程管理的参考模型了,而是某个特定软件组织为了迎合或者满足 CMM/CMMI 评估的需要所定义出来的某个具体软件过程.显然,将这个为了特定目的而定义出来的软件过程的缺点视作 CMM/CMMI 模型的缺点是不合适的.

关于软件过程管理,还需说明一点,按照 Humphrey 在《Managing the Software Process》[19]一书中对软件过程管理这一术语的解释来看,所谓“软件过程管理”应该同时包含按照既定过程的执行和不断提升过程能力两个方面的内容.因此,软件过程改进应被视作软件过程管理的一部分.其常用参考模型就是 PDCA(plan-

do-check-act)[20]和 IDEAL(initiating-diagnosing-establishing-acting-leveraging)[21]这两个过程改进的元模型.但是应该注意的是,软件过程改进不能脱离现有的过程基础,否则,过程改进就会成为无根之木.实践中,软件过程管理和软件过程改进两者不能割裂,既不能脱离改进谈管理,也不能脱离管理谈改进.因此,像 CMMI 和 SPICE[22]这样的模型,软件过程管理参考模型和软件过程改进参考模型两种说法均可.

1.3 软件过程的一种多维视角

实践中,越来越多的人开始意识到:一个合适的软件过程往往需要通过过程组合和裁剪,将多个不同类型的软件过程的优势整合起来.例如:Boehm 等人提出一种策略来平衡敏捷和规范(discipline),从而发挥两种类型的软件过程的优势[23,24];Kuhrmann 等人的多项工作都关注在影响软件过程组合、裁剪和定制的项目上下文(context)特征上,试图为软件过程的组合提供参考[25−27].

显然,为了对过程进行合理的组合和裁剪,必须首先建立起对软件过程特点的正确理解.目前讨论比较多的是将软件过程分为敏捷和非敏捷两类[23,28],同时认为,这两类过程各具优势,有着不同的适用场景.Boehm 等人以“猴”(不能长途负重,但是可以采摘高处水果,比喻敏捷过程)与“象”(适合长途负重,不能采摘高处水果,比喻非敏捷过程)形象地比喻上述两类过程[23],认为有必要通过组合和裁剪来融合这两种类型的软件过程,发挥各自的优势[23,28].这种方式的风险在于:单一维度往往并不足以描述一个软件过程的全部特征,因而限制了对软件过程的全面理解,进而影响了软件过程元素的合理选择和组合.例如,尽管同属于敏捷阵营,XP 方法和 SCRUM 方法关注的重点完全不同:前者关注工程实践,而后者则更多的是一个包含很多管理实践的过程框架.为了更加清楚地展示软件过程多维度特征的概念,我们以一个立方体(即 3 个维度)来刻画软件过程特征(如图 3 所示),下面分别加以阐述.

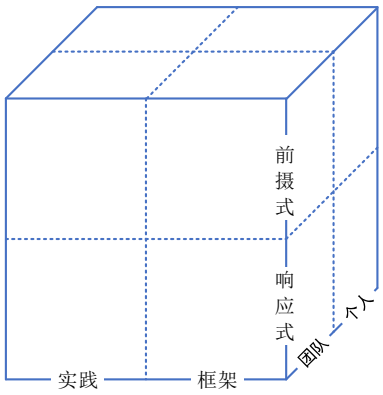

Fig.3 A three-dimensional perspective to characterize software processes

图 3 刻画软件过程特征的三维视角

- 过程实践(practice)vs.过程框架(framework)

在各种不同的软件过程中,我们发现有些过程关注的重点是具体的实践,而有的过程则提供了相对抽象的整体化管理框架.例如,极限编程 XP 是一个关注具体实践的软件开发方法.典型地,在 XP 方法中包括了像计划游戏(planning game)、重构(refactoring)、结对编程(pair programming)以及持续集成(continuous integration)等典型的软件工程实践.这些具体实践的相应描述为软件开发提供了有用的指南.但是,如何来组织整个项目团队以实现项目的目标,仅仅有这些软件工程实践还不够,还需要有一个整体的过程框架将这些工程实践装入其间.例如,SCRUM 方法就是一种非常典型的过程框架.SCRUM 方法中的典型实践包括计划游戏、产品 backlog、固定的 timebox、短迭代周期的 sprint、每日站立式会议、燃尽图等.这些实践以管理实践为主,整个过程框架则提供了项目的全景图.显然,一个可以工作的软件过程应当既提供整体观(即过程框架),让过程使用者(即工程师)形成统一的项目愿景;同时也应该提供非常具体的实践指南,让过程使用者知道该如何完成每一项关键任务.因

此,实践中我们可以看到,XP 和 SCRUM 整合是非常常见的做法[29].此外,作为侧重提供管理类实践指导的 PSP (personal software process)[30]和 TSP(team software process)[31]整合,也一定程度地体现了实际可工作的软件过程应当兼有过程框架和具体实践这一要求.

- 团队(team) vs. 个人(individual)

现代软件开发一般都会以团队形式展开,以期缩短从需求到上线之间的时间.因此,大部分软件过程关注的都是团队这一层级.相对而言,只有极少数软件过程关注个体层次,例如 PSP 过程和 Cleanroom 方法.然而值得注意的是:即便软件项目的开发是以团队形式进行,在整个过程中,大部分的工作都是软件工程师以个人工作的方式进行的,例如详细设计、编码以及单元测试等等.软件项目的进度其实是取决于参与其间的每一位软件工程师个体的工作进度,软件系统的质量亦如此.从这个意义上说,指导和规范个体软件工程师工作的软件过程同样是不可或缺的.Turk 等人也指出:“团队成员应该具备足够的经验和技能来定义和改进软件,从而实现敏捷过程.这就要求软件团队应当由聪明、能干和经验丰富的人来组成.”[32].显然,软件工程师不是天生具备上述技能的,因此软件过程也应当关注个体软件工程师,指导他们工作,帮助他们成长.

- 反应式(reactive) vs. 前摄式(proactive)

这个维度描述的是在应对软件开发过程中各种变更时的处理方式.顾名思义:反应式指的是在变更发生之后再应对,而前摄式则是在变更发生之前采取措施来减少发生变更的可能.更一般化地,我们也可以把变更推广为软件项目进展过程中的各类问题.相应地,反应式软件过程和前摄式软件过程则代表了两种不同的项目管理策略.大部分敏捷(agile)方法都可以归为反应式软件过程.我们使用“反应式”和“前摄式”的主要目的是为了打破敏捷方法和非敏捷方法之间的传统界限,便于过程的组合(process composition)和定制.例如,软件重构(refactoring)尽管是一个典型的敏捷实践,但是大部分情况下,软件重构的目的是为了改善代码结构,以应对未来可能的变更,减少返工代价.从这个意义上说,软件重构应该归为具有前摄特征的一个软件工程实践.只有打破上述传统界限,才能使得在软件过程的组合和定制时具有足够的灵活性.

按照多维度的方法来刻画软件过程特征,XP 方法可以用三元组(practice,team,reactive)来表示.类似地,TSP 则可以表述为(framework,team,proactive).显然,我们还加入了更多正交的维度进一步刻画软件过程的特征,为系统化的软件过程组合提供条件[33].我们以 XP、SCRUM、PSP 以及 TSP 为例(见表 1)来解释软件过程多维视角下的过程组合方式.例如:前文已经讨论过,一个好的过程应该既提供整体观(framework),也提供具体实践指导(practice)[27,33].从这个意义上看,XP 和 SCRUM 组合、PSP 和 TSP 组合、PSP 和 SCRUM 的组合以及 XP 与 TSP 的组合都是比较合理的过程组合方式;相反地,XP 和 PSP 的组合以及 SCRUM 和 TSP 的组合则不大合理.在主流科研文献搜索引擎上的检索结果也表明,前者的组合方式(即 practice 和 framework 的组合)的相关研究显著多于后者.此外,按照 Turk 等人的提法[32],软件过程还应该为个体软件工程师提供实践指导.从这个意义上说,除了 PSP 和 TSP 的天然组合之外,将 PSP 融入到其他软件过程的做法也值得研究.事实上,研究者已经开展了一些相关工作[34−37].相比较于单一维度(即敏捷与非敏捷),多维度视角下的过程特征识别和过程组合显然具有更好的针对性和可操作性.

Table 1 Characterization and composition of software process

表 1 软件过程的特征刻画和组合

	Practice	Framework
Reactive	XP	SCRUM
Proactive	PSP	TSP

2 软件过程发展史

伴随着软件规模从数十行到数千万行(甚至接近不可计数(部分原因是现代软件系统的边界已经越来越模糊,例如,基于互联网的很多系统都是由大量的软件服务(子系统)组合而成,很难定义边界))这样的发展历程,其相应的软件过程的演化也成为一段波澜壮阔的发展史.期间出现的构成体系的软件过程和方法有数十种,如果加上这些过程的一些变化,更是不计其数.正如前文讨论的那样,软件项目管理是为了确保项目目标的达成.那

么相应地,软件过程则是软件项目管理经验的汇总,成为后续项目的参考,更好地支持项目管理目标的实现.因此,探讨一下软件项目目标达成所面临的困难和挑战,对于我们理解不同的软件过程是有益的.Brooks 在《没有银弹》一文中总结了软件开发的四大本质(essential)难题,即复杂度、一致性、可变性以及不可见性[38],一直被业界广为认同.严格来说,只有不可见性是软件的固有属性,不会因为软件项目的上下文环境的不同而发生变化;另外 3 个本质困难都会随着软件项目上下文环境的不同而变化.这 4 个本质难题对软件项目管理带来的挑战往往相互促进,例如:随着软件开发的复杂度的提升,一致性、可变性以及不可见性所带来的对软件开发的负面影响也会不断增加.在软件发展的不同历史阶段,由于技术发展、应用领域以及商业环境等因素的影响,除了不可见性之外,另外 3 个不同的本质难题的凸显程度往往不一样.例如,尽管互联网时代的软件系统比以往更为复杂,但是,由于要服务海量的最终用户,软件系统的可变性(需求变更)和一致性(快速演化)的问题反而更加突出.为了适应这种差异和变化,不同历史时代的软件过程也往往呈现了不同的特征.我们以软件发展的 3 个主要历史阶段为主线,梳理软件过程的演化历史,从中我们可以更加清楚地观察到这一现象.

2.1 软/硬件一体化(20世纪50年代~70年代中叶)

这是计算机软件从萌芽状态慢慢发展的一段时间.总体上,软件是以硬件附属品的形式存在,而不会单独存在.当然,随着技术的发展和人们观念的转变,软件的地位也一直在上升.图 4 描述了在企业 IT 方面的支出中,软件成本和硬件成本比例的变化趋势.如果我们假定“摩尔定律”对硬件和软件的影响类似,那么我们可以明显地看到,软件的地位在不断上升.大体上,软/硬件一体化阶段依据软件在一个计算机系统中的比例可以细分为如下两个阶段.

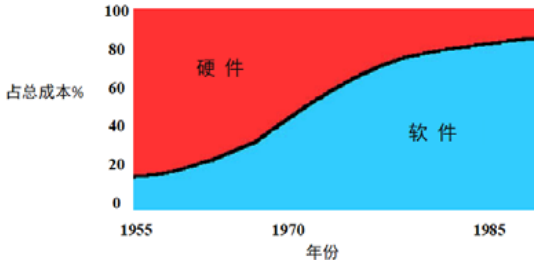

Fig.4 Large organization hardware-software cost trends[39]

图 4 大型企业中软/硬件成本变化趋势[39]

- 软件完全依附于硬件阶段

软件在这个阶段完全处于萌芽状态.有一些主要的特征决定了这个时代的软件过程是非常接近硬件开发流程的.例如:软件往往是用来支持硬件完成一些计算任务,其功能相对单一,复杂度有限,几乎也不需要考虑软件的需求变更;其次,作为计算资源的硬件,其每小时使用成本远大于支付给软件开发者的报酬;最后,项目团队往往也是以硬件工程师和数学家为主,软件工程师作为一种职业同样是萌芽状态[40].相应地,这个阶段的软件过程有着非常明显的硬件工程过程特征,例如,往往采用线性的类似于瀑布模型的流程;同时,一些典型的硬件工程口号得到推崇,例如“measure twice, cut once”.对应到软件领域,那就是对所编写的代码进行反复的多次检查,确保无误后,再上机计算.图 5 是这个阶段非常典型的软件过程,我们可以看到线性顺序的过程结构和大量规格定义的活动.

- 软件作坊阶段

随着技术的进步,人们对计算机系统的认识加深,软件应用领域也得以扩展.相应地,社会对软件人员数量的需求也随之增加,很多非软件领域的人员涌入了软件开发领域.所以,整个 20 世纪 60 年代的软件开发极具软件作坊特征.当然,还有其他一些原因,使得这个时代的软件过程具有轻量级的特征:首先,大部分软件功能仍然比较简单,规模不大,复杂度有限,因此即便出错,其错误修改代价也不大;其次,出现了一些更好的软件开发基础

设施(例如高级程序语言),使得快速完成功能的编码并且调试修改成为可能.此外,质疑权威的文化盛行,导致了牛仔式的个人英雄主义产生.上述这些因素,使得整个 20 世纪 60 年代最盛行的软件过程就是所谓的“编码加改错(code and fix)”方式.尽管这种方式在现代软件工程当中被认为是糟糕的做法,但从当时的角度来看,仍然不失为一种适应软件需求时代特征的合理做法.当然,即便在 20 世纪 60 年代,也并不是所有的软件都是规模小且复杂度低的软件.因此,当“编码加改错”这样的软件过程应用在相对复杂的软件开发中时,其问题也就出现了.除了极少数成功案例(如 IBM 的 OS-360)之外,大量的有一定规模的软件项目都失败了,直接导致了在 20 世纪 60 年代末 NATO 科学委员会召集两次会议[41,42]讨论“软件危机”,并且提出了“软件工程”的概念.

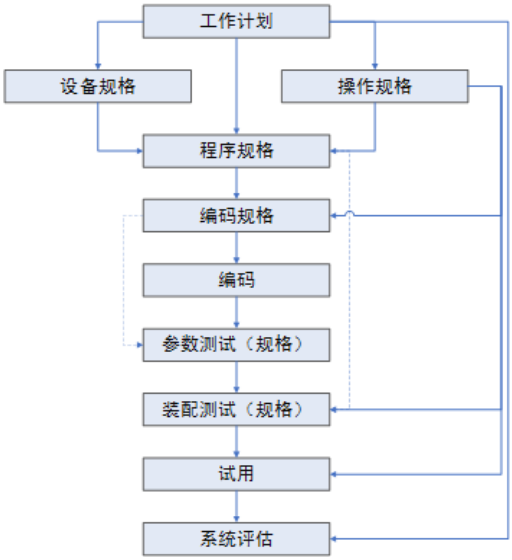

Fig.5 Software development process for SAGE which is similar to hardware development process[43]

图 5 类似硬件过程的 SAGE 软件开发过程[43]

2.2 软件成为独立的产品(20世纪70年代~90年代)

尽管以盈利为目的的软件公司在 20 世纪 60 年代已经出现,但是真正发展起来的软件公司还是出现在 20 世纪 70 年代之后.很明显的特征就是软件作为独立的产品出现了.由于摆脱了硬件束缚,同时也受到用户期望的影响,软件的功能越来越强大,相应的软件系统规模和复杂度也显著增加.此外,由于软件开始服务于更多领域的用户,尤其是 20 世纪 80 年代出现了个人计算机之后,普通人作为软件的主要用户开始出现在历史舞台.这直接导致除了复杂度和不可见性之外,软件功能需求的多样化(以及变更)以及软件产品兼容性要求(一致性)等开始变得重要起来.某种角度上说,Brooks 总结的软件开发的本质难题在这一阶段才真正引起整个行业的广泛认同.结合前文提及的软件“摩尔定律”,商业公司所面临的挑战不仅仅是上述这些软件开发本质难题,还有时间上的压力,即:需要尽可能快速地将软件产品推向市场,以期获得较高的收益.

很明显,盛行于 20 世纪 60 年代的牛仔式开发方法已经不能适应这个时代的软件开发了.作为矫枉措施,两种软件开发方法在这个阶段的初期获得了重视:其一是形式化方法[44,45],该方法试图通过数学手段证明程序的正确性,从而确保软件产品质量;其二是自顶向下的结构化方法以及相应的瀑布生命周期模型[46].这其中,瀑布模型值得解释一下.事实上,这也是业界对软件过程误解最多的概念之一.如图 6 所示,这是一个典型的瀑布模型,其特征是,基本上顺序的开发流程以及在每一个阶段采用先尝试后展开(即 build it twice)的开发策略.如果仔细阅读 Royce 论述瀑布模型的论文[46],我们会有如下发现.

- 首先,Royce 提出的瀑布模型并不像大部分文献中提及的简单顺序模型(在图 6 中去掉左下角的尝试过程以及每个阶段的回溯步骤);

- 其次,Royce 提出的瀑布模型也不是单一模型,事实上,他提出的是从先设计再开发、设计文档化、“Build it twice”、规划和监控测试以及引入客户这样一个软件开发过程的改进路径;并且,他认为,随着过程元素的增加,开发成功可能性会提升,但是开发成本也会上升,因此,开发团队应该做好两者之间的平衡;
- 最后,也不像很多文献中提及的那样,Royce 本人对这样的生命周期模型在大型软件开发中持否定态度.事实上,他认为采用这样的一个软件过程是非常有必要的,尤其是考虑到大型软件开发往往比预想的要更为复杂,因此,不能用过于简单的软件开发方法来应对.非常可惜的是,他的建议并没有引起足够的重视,直接导致了一个概念简单、但是官僚死板的瀑布模型(DoD-Std-2167)[47]被广为传播,并且成为后来大部分“新”型软件开发过程的拥趸对比和攻击的对象.

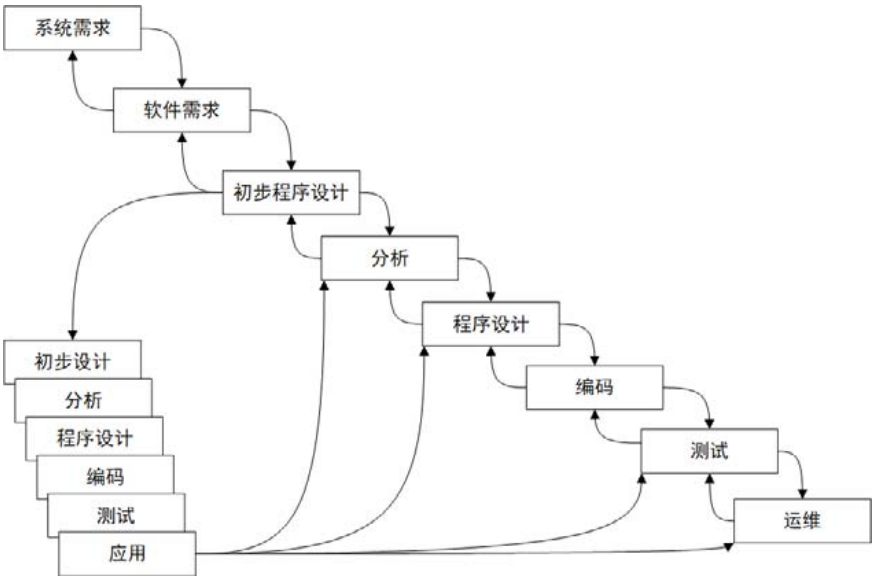

Fig.6 Royce waterfall model[32]

图 6 Royce 提出的瀑布模型[32]

形式化方法在扩展性和可用性方面存在不足,而顺序瀑布模型则被解读成一个重文档、慢节奏的开发过程.因此进入到 20 世纪 80 年代,如何提升生产效率和开发质量成为业界最为关心的话题.从软件开发方法的角度看,80 年代乏善可陈.然而在这个时代出现了一个影响至今的软件开发技术,即面向对象技术.面向对象技术不仅有效缓解了复杂度、不可见性、一致性和可变性等软件开发本质难题给软件开发带来的困难和挑战,也直接影响了软件开发过程.例如,RUP(rational unified process)过程[48,49]就是一个典型的例子.仔细研究 RUP 过程,我们可以发现,它的流程定义、概念和术语、相关支持工具,几乎都是围绕着面向对象技术而展开的.面向对象技术的一些衍生技术,例如面向方面编程 AOP(aspect oriented programming)、领域驱动设计 DDD(domain-driven design)等,则在后来的软件开发方法中也被广为应用.20 世纪 80 年代出现的另一个具有标志性意义的工作是以 CMM 为代表的软件过程改进参考模型,由于 CMM 不是一种软件过程,这在前文已经有很多讨论,这里不再赘述.

尽管没有直接体现在软件过程和开发方法的变化上,20 世纪 80 年代的技术进步还是为 20 世纪 90 年代大量过程和方法的涌现奠定了基础.当然,这一切的背后离不开软件发展的新动向.

2.3 网络化和服务化(20世纪90年代中期迄今)

进入 20 世纪 90 年代,人们对软件的使用方式出现了一些变化:一方面,作为独立于硬件存在的软件,其功能继续复杂化,规模继续扩大(软件“摩尔定律”);其次,软件的用户数量继续增加,尤其到了后期,随着互联网和移动互联网时代的到来,一款软件系统或者线上服务动辄拥有百万级以上的用户量.同时,由于竞争的需要,需求不

确定性和系统的快速演化成为一个日益突出的问题.最后,软件分发和使用方式也出现了显著的变化,从典型的光盘复制逐渐过渡到基于网络的服务形式,即 SaaS(software as a service)[50],使得软件系统的版本更迭时间有了大为缩短的潜力.这些新情况的出现,使得软件开发的四大本质难题中,可变性和一致性对软件开发的影响更为突出,因而也催生了整个软件过程历史上最为纷繁的一个时代,大量的软件过程在这个阶段涌现出来.其中,具有迭代式开发特征的一系列软件开发方法逐渐得到软件行业的广泛接受和应用.迭代式开发历史悠久,其出现时间甚至可以前推至 20 世纪 50 年代[51].迭代式开发不适合视为一种具体的开发过程,例如,产生于 20 世纪 80 年代的增量式模型[51]、螺旋式模型[52]以及原型法[53]等均具有迭代式开发特征.尽管在细节上有些小的差别,但其本质都是类似的,即:将一个大型软件系统的开发过程视作一个逐步学习和交流的过程,软件系统的交付不是一次完成的,而是通过多个迭代周期,逐步来完成交付.显然,这种方式与之前流行的单一顺序过程有着显著差异.如果说在此之前,迭代式开发只是软件开发方法的选项之一,在软件发展进入网络化和服务化阶段,具有这种特征的软件过程得到业界的空前重视,演化出了一系列新方法.这些新的软件开发方法在项目实践中逐渐形成体系,终于在 2001 年的“雪鸟会议”上,17 位软件工程实践者达成共识,以“敏捷”一词来描述这些软件过程的主要特征[54].

- 敏捷软件开发方法

敏捷软件开发(agile software development)是一组强调在不确定和混乱的情况下适应软件需求快速变化的、基于迭代式开发的软件开发方法和实践.当前,敏捷方法是一种主流软件开发方法,广泛应用于各种软件开发中,也包括很多规模较大的软件开发中.

敏捷软件开发主要由有经验的软件工程师和咨询师提出,而非来自于学术界的研究成果.在 2001 年“雪鸟会议”期间,与会者形成了一些关于软件开发的共同观点,即敏捷宣言[54]:“个体和互动胜过流程和工具;可以工作的软件胜过详尽的文档;客户合作胜过合同谈判;响应变化胜过遵循计划”.该宣言定义了敏捷软件开发的核心价值观和原则,而这些原则具体是由敏捷实践以及敏捷实践的组合——敏捷方法来实现的.

敏捷体系的一种比较公认的解构方式是按 5 个层次划分,分别是价值观、原则、方法、实践和工具(如图 7 所示).越是高层,其概念就越抽象,相应地,其内容也就越少.因此,我们可以用一个金字塔形状来形象地描述敏捷体系的 5 个层次.

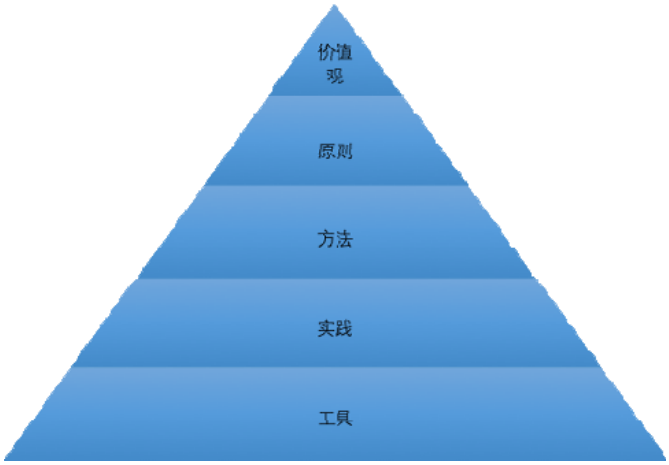

Fig.7 Five-Layers of agile method

图 7 敏捷的 5 个层面

随着敏捷软件开发的发展,敏捷方法的数量逐渐增多,目前大约存在 20 种左右的敏捷方法,这其中最受关注的敏捷方法主要有 SCRUM、XP 和 Kanban[29,55],下面分别简单加以介绍.

- SCRUM:着重于项目管理,尤其适用于难以提前完全规划好的项目.

SCRUM 以经验过程控制理论为依据,采用迭代、增量的方法来提高产品开发的可预见性并控制风险.SCRUM 框架包括一组 SCRUM 团队和与其相关的事务.SCRUM 团队以自组织、跨职能和迭代方式工作.典型地,一个 SCRUM 团队有 3 个角色:(1) SCRUM Master,负责确保成员都能理解并遵循过程;(2) 产品负责人,负责最大化 SCRUM 团队的工作价值;(3) 团队,负责具体工作.SCRUM 利用时间盒实现规律性.SCRUM 中的 Sprint 是贯穿于开发工作中保持不变的一个月(或更短时间)迭代.所有的 Sprint 都采用相同的 SCRUM 框架,并且都交付潜在可发布的最终产品增量.

- XP(extreme programming):极限编程着重于软件开发的最佳实践.

早期的 XP 包含 12 个实践:计划游戏、短周期迭代、隐喻、简单设计、测试、重构、结对编程、代码集体所有制、持续集成、40 小时工作制、在线客户、编码标准.2004 年修改后的 XP 引入了更多的实践,同时将这些实践区分为基本实践(例如坐到一起、完整团队、结对编程等)和扩展实践(例如真实客户参与、增量部署、每日部署等).XP 方法也有一个迭代式的过程框架,用来组合上述实践.但是 XP 过程框架中的管理实践较少,后来逐渐被 SCRUM 取代.SCRUM 和 XP 的组合,一度也成为敏捷开发的标准配置.

- Kanban:Kanban(看板)是日语单词,是“可视卡片”(或标志)的意思.

在丰田公司,看板专指将整个精益生产系统连接在一起的可视化物理信号系统.看板的主要规则有:

(1) 可视化工作流:把产品切分成小块,将每一块写在一张卡片上,然后将卡片贴到墙上;墙上的每一栏都有名称,以此显示每张卡片在工作流中所处的位置;

(2) 限定在制品 WIP(work in progress):针对工作流的每个状态,明确限定正在进行中的工作项数量;

(3) 衡量并管理周期时间:完成一个工作项的平均时间,有时称为前置时间(更贴切的术语可能应该是流通时间).优化流程,让周期时间尽可能短并尽可能可预测.

Pekka 等人[56]分析了各种敏捷方法,并给出了它们之间的演化关系(由于此文发表于 2003 年,因此没有包含看板方法和精益软件开发方法),如图 8 所示.

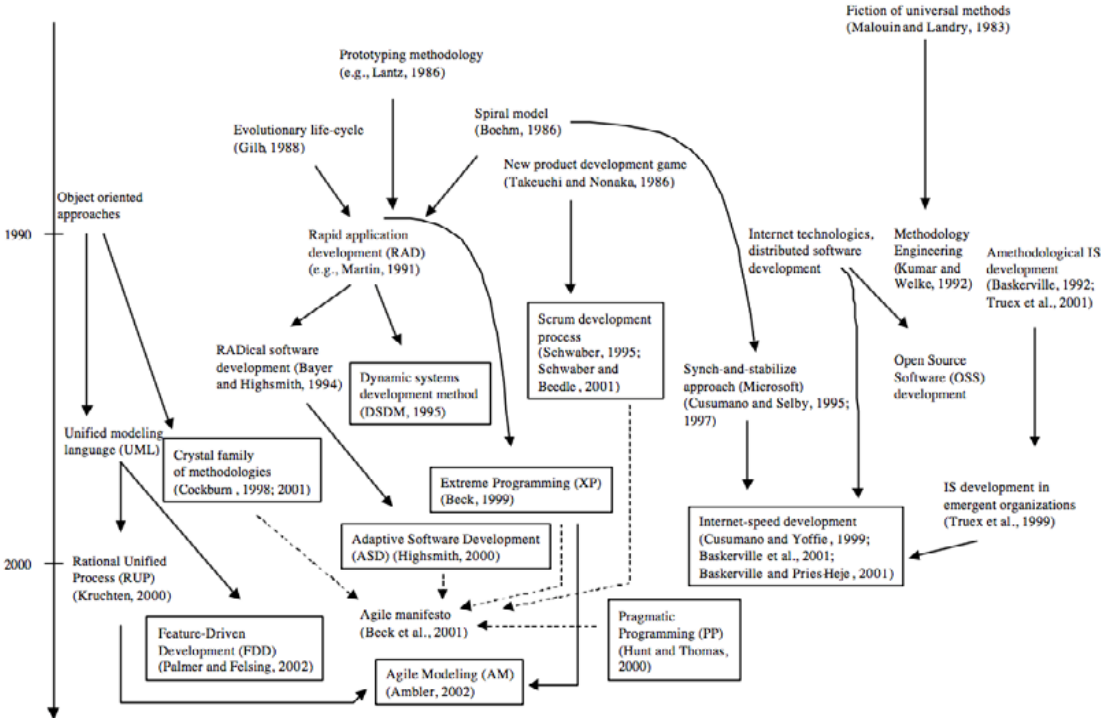

Fig 8 Evolution of agile method[56]

图 8 敏捷方法演化图[56]

如果以 2001 年敏捷宣言的提出作为敏捷方法的里程碑事件,我们可以发现:敏捷方法往往都从一些早期软件工程实践演化而来,而不是凭空出现.此外,这些方法出现的时间也较为集中,基本符合前文所述软件逐渐走向网络化和服务化的这一发展阶段.换句话说,也正是这一发展阶段,软件的特点使得具有类似特点的一系列软件过程被大量地应用在当时的各个软件项目开发当中.

伴随着敏捷软件开发的发展,关于敏捷软件开发方法和之前的传统软件方法(原文使用“传统”一词来描述单一顺序过程,我们并不完全认同用这个词汇)的争议也越来越多.Nerur 给出了两者的区别(见表 2)[57].

Table 2 A comparision between traditional software development and agile development[57]

表 2 传统软件开发与敏捷软件开发的对比[57]

	传统软件开发	敏捷软件开发
基础假定	可以完整描述系统需求,系统是可预测的,通过周密、完整的计划来开发软件	基于需求变更和快速反馈,通过小团队持续设计、测试、改进完成高质量的软件开发
管理风格	命令和控制	领导力和合作
知识管理	显式	非正式
交流	正式	非正式
开发模型	生命周期模型(瀑布或其他变种)	演进-交付模型
组织结构	机械的(正式的)大型组织	灵活的小型组织
质量控制	严格的计划和控制,后期大量测试	持续演进需求、设计、方案.持续测试

Version One 公司每年都会进行一次全球敏捷开发的调查,目前已经进行了 11 次.虽然这不是严格意义上的学术研究,但在工业界和学术界的影响力都非常巨大,是持续时间最长、参与人数最多的敏捷调查.2018 年发布的第 11 次敏捷状态调查报告指出,94%的受调查者所在的企业使用敏捷方法.报告宣称的敏捷带来的益处排在前 3 位的是:(1) 88%的被调查者认为,敏捷帮助他们获得了管理变更优先级的能力;(2) 83%的被调查者认为提高了项目的可视化;(3) 83%的被调查者认为提高了团队生产力.对于具体的敏捷方法,排在前 3 位的方法分别是 SCRUM(58%)、SCRUM/XP 整合(10%)和 Custom Hybrid 自定义(8%)[58].

敏捷方法作为与基于计划的传统软件开发方法相对的软件工程方法,从诞生开始就面对着众多的争议.很多实践者往往从个人的经验和观点出发进行争论,缺乏严谨的科学基础.也有很多学者和实践者并不认同敏捷软件开发,他们认为,没有足够的科学证据支持敏捷所宣称的益处.对此,许多研究者对敏捷软件开发开展了大量实证研究来确定敏捷软件开发的有效性和局限性,同时也展示了当前敏捷软件开发在工业界的使用情况供实践者参考.

- 开源软件开发方法

开源软件(open source software)是一种源代码可以任意获取的计算机软件.开源软件开发方法是伴随着互联网在全球的快速发展而兴起的.以 Linux 为例,Linux 起步发展的时间(1993 年~1994 年)与 Internet 在全球的快速发展几乎是同步的.严格来说,开源软件开发方法的说法略显牵强,更为确切的说法应该是一种基于并行开发模式的软件开发的组织与管理方式.这种方法依赖于分散在全球的开发者和使用者的协作,而只有 Internet 才能为这种大规模协助提供交流沟通的工具.廉价的 Internet 是 Linux 模式得以发展的必要条件.

在缺乏自顶向下的严格项目管理和传统上大家认可的软件工程最佳实践的情况下,开源软件开发方法取得了巨大的成功,Eric Steven Raymond 在《大教堂与集市》[59]一文中对开源软件开发方法进行了深入分析.Eric 提出了“Linus 定律”:“如果有足够多的 beta 测试者和合作开发者,几乎所有问题都会很快显现,然后自然有人会把它解决.”同时,Eric 给出了一些具有指导意义的实践,例如“早发布,常发布,倾听用户的反馈”“把你的用户当成开发合作者对待,如果想让代码质量快速提升并有效排错,这是最省心的途径”“设计上的完美不是没有东西可以再加,而是没有东西可以再减”等,也得到了很多软件开发者的接受和认同.

作为一种软件开发方法,开源软件开发的贡献者(contributor)缺乏严格的人员和任务上的管理.为了保证开源软件的质量,开源社区一般都有严格的代码提交社区审核制度.开发者通常一次只允许提交少量代码.代码提交后会触发社区代码审核,经过开发社区的主要贡献者(committer 或 maintainer)确认后才能够通过代码配置系统整合到代码主分支上.

随着开源软件的快速发展,众多软件企业开始使用开源软件或者将开源软件集成到自己的商业软件中.小到个别功能模块,大到整个 IT 基础设施,我们都可以看到各种开源软件的存在.同时,众多软件企业开始投入资源参与社区开源软件的开发.比如:Intel 公司贡献了大量 Linux 核心代码;IBM 贡献了 Eclipse;Google、Facebook、Netflix 等公司均将公司内部很多项目开源,一方面为软件社区提供了优质的软件,同时也加速了公司自身的软件开发速度和质量.不仅如此,不少大型软件公司也开始尝试将开源软件开发方法引入到内部的商业软件开发中,提出了所谓的"内部开源(inner source)"方法[60],也获得了很好的效果.

此外,开源软件开发方法还启发了一种称之为众包(crowdsourcing)[61]的软件开发组织方式.众包的方式继承了开源软件开发的大部分实践,但与传统开源软件开发方法还存在一些差异.例如:在众包方式下,往往是由确定的管理者(个人、小组或者公司)来主导软件开发的内容和软件系统的演化走向,管理者发布开发任务,而贡献者往往会因为完成开发任务而获得一定的报酬(类似一种雇佣关系);而在传统的开源软件开发中,贡献者与开源系统的管理者之间是一种相对对等的关系,管理者一般不会向社区发布明确的开发任务,而贡献者更多的是以服务自己和社区的目的来完成开发工作,一般都是无偿的.

- DevOps

进入 2000 年,随着互联网应用的日益普及,用户对软件系统和服务提出了更多的要求,"多快好省"已成为大多数互联网时代软件用户的基本期望.具体而言,功能要丰富、更新要及时、要稳定可靠同时用户获取服务的成本不能过高.然而,有一些因素却制约着这样的目标的达成.

- 首先,由于互联网时代的用户已经经历了较为长期的信息时代的熏陶,对于软件产品和服务已有较为深入的理解;而作为软件产品和服务的提供方,为了吸引用户和流量,往往都提出了以用户为中心的口号,综合起来,用户的需求多样性问题更加突出;
- 其次,软件产品和服务的地位已经从辅助社会生活发展到不可或缺的状态;服务宕机已经越来越成为一种难以接受的情形;
- 最后,长期信息化建设的结果使得软件产品或者服务的部署环境错综复杂,软件系统从通过测试到在生产环境部署和应用之间的鸿沟日益凸显.

所有这一切,都使得人们对软件产品和服务的需求与软件产品和服务的开发能力之间越来越不匹配(请注意,这正是软件危机的起源).DevOps[62,63]这种方法应运而生.从起源来看,DevOps 是敏捷开发的延续,它将敏捷的思想延伸至运维(operation)阶段,以期快速响应变化和交付价值.目前,DevOps 方法已对软件产业产生了深远影响,越来越多的软件企业开始采用这种新的模式.DevOps 之所以产生如此巨大的影响,我们认为这不是偶然的.这种方法本身具有的特性非常适合在:(1) 需求很难确定;(2) 需要快速响应变更;(3) 需要快速提供价值; (4) 需要高可靠性、安全性等这样的所谓互联网时代软件环境中得到应用.限于篇幅,完整阐述 DevOps 方法不大可能,我们简单地对若干 DevOps 典型实践和技术进行总结,从中可以大致理解 DevOps 是如何解决前文提及的"多快好省"的问题的.

- 首先,DevOps 的方法论基础是敏捷软件开发、精益思想以及看板 Kanban 方法.这是一种鼓励快速迭代、同时尽快向用户交付价值的软件过程.用户往往在项目早期就可以使用软件系统的部分功能,并在其后持续获得系统的更新和升级;
- 其次,以领域驱动设计为指导的微服务架构方式,帮助用户重塑和分解组织业务架构,解耦复杂应用系统,从而支持不中断服务的系统更新;
- 第三,大量虚拟化技术的使用,使得开发测试环境与生产环境几乎没有差别,新的功能模块(或者服务)封装在容器中,以类似集装箱形式快速部署到生产环境中;
- 第四,一切皆服务 XaaS(*X* as a service)的理念指导,不仅大大降低了开发、部署和运维的难度,同时也降低了使用成本.甚至一些云服务提供商提出了函数即服务 FaaS(function as a service)[64,65]的概念,按照函数执行期间所需资源计费,极大地提升了计算机关键资源(例如 CPU、网络等)空闲时间的利用率,从而大大降低了使用成本;

- 第五,构建了强大的工具链,支持高水平自动化.代码从编写完成到部署上线,几乎不需要人工干涉.

上述这些实践不是各自独立的,而是需要相互配合以支持前文提及的软件开发目标.从一些行业调查数据来看,我们观察到了一些在以往难以想象的提升.比如,在不中断服务的情况下,Google 公司的服务更新达到了每天 5 000 次,Amazon 的更新则达到了 23 000 次以上[66].作为一种新的软件开发和运维方法,DevOps 在工业界已有大量的应用和探索.相对而言,学术界对 DevOps 的相关研究起步较晚,对这种方法的局限性的认知还不够系统和全面.

3 发现和反思

综合前文的梳理和讨论,我们总结若干发现如下,试图为软件组织与管理方法在软件过程元素的选择、组合以及定制方面提供一些参考.

发现 1. Brooks 在《没有银弹》一文中总结的软件开发四大本质难题,跨越了时代但仍然具有启示意义.

在很多文献中,会使用“传统”一词来描述历史上的一些重型方法(尤其是瀑布方法),用以区分敏捷方法.事实上,作为敏捷方法基石之一的迭代式开发方法或者增量式开发方法起源于 20 世纪 50 年代,兴盛于 20 世纪 70 年代,远远早于敏捷方法被正式提出的年代[51].此外,作为敏捷方法中极为关键的一种实践,测试驱动开发(test driven development),其前身 Test First Development 在水星计划(project mercury)中已开始应用,而其时代则可追溯到 20 世纪 60 年代.我们可以看到:软件过程和管理方法试图要解决的问题,过去发生了,现在还在发生着,未来还会继续存在.正如 Brooks 所总结的那样,这是软件开发中的一些本质(essential)的挑战和困难.这一方面暗示着我们不大可能找到有效的方法来彻底解决这些问题;而另一方面也表明,我们在探索过程中逐渐积累的经验也不会因为时代的变迁而消亡.

发现 2. 管理方法和过程不会独立于技术而存在.

从面向对象技术的产生,并且由此引发了 20 世纪 90 年代软件开发方法的大量涌现,我们就可以观察到这样的现象.正如软件过程的定义(如图 2 所示),广义过程本来就应该包含技术元素在里面.另外一个典型的例子是 DevOps 方法,如果没有微服务架构设计和云计算来支撑,软件系统功能的高质量和高频交付是难以想象的.从这个意义上说,如何将管理实践和技术实践有机融合在一个软件过程中,应成为在软件过程的组合和定制时重点考虑的内容.

发现 3. 人的因素还是居于主导地位.

这一点是毋庸置疑的.软件开发本质上看是智力劳动,因此整个开发过程中没有办法将人的因素完全排除掉.事实上,在广义软件过程中,人员也是其中的一极.比较公认的开始重视人在软件开发中的影响以两本书为代表,即《程序开发心理学》[67]和《人件》[68].其实,现代软件工程方法都非常重视人的因素,几乎所有现代流行的软件工程方法都鼓励自主方式(self-directed)的软件项目开发[9,30].

发现 4. 相比较于软件的发展,软件过程具有滞后性.

这个原因易于理解,因为软件过程本身就是经验教训的总结.往往都是在现实中出现软件开发能力不能支持人们对软件需求和期望这样的问题之后,相应的软件过程和开发方法才会开始演进,以解决上述问题.从这个意义上说,除非要解决的问题出现显著变化,否则,对软件过程的研究和探索的重点则仍然应该放在如何更好地使用现有过程和方法上,而不必去创造新的方法.

发现 5. 迭代式方式将成为主流方法.

受到应用大环境的影响,当前软件开发所要解决的问题尽管仍然可以归结为经典的 4 个本质难题,但在程度上与过往相比已经有了翻天覆地的变化.某种角度上来说,单一的顺序型开发过程很难在当前的软件开发过程中得到应用,迭代式开发方法已经成为一种主流方法,甚至在未来较长时间内,应该都是主流方法.这种方式鼓励的快速交付和反馈,使得用户、开发者以及作为连接桥梁的软件系统三者之间的互动更加频繁,用户和开发者对当前所需解决的问题以及未来演变的理解可以快速取得一致.显然,这非常适合于当前越来越趋向于网络化和服务化的软件系统开发和使用的发展趋势.

发现 6. 反应式开发最终应向前摄式发展.

软件开发尤其是现代软件开发面临着很多不确定性,提升应变(reactive)能力是非常必要的.尽管如此,从控制风险、提升质量、降低返工成本等因素出发,软件过程仍然应该引入更多具有前瞻(proactive)特性的过程元素,在不确定中找到确定性因素,并进而强化确定性,消除不确定性,应该是一种基本的软件项目开发策略.事实上,我们前文已经讨论过,原型法、螺旋式开发以及以迭代式作为基础模型的大量敏捷方法都将软件开发过程当成是一个逐步学习和交流的过程,在这个过程中,开发者和用户对软件系统的理解越来越趋向一致,从而消除很多不确定因素.从这个意义上说,都是反应式开发发展成前摄式开发的策略的体现.

发现 7. 简洁、直观的过程.

我们发现,能够让开发者广为认同和使用的软件过程几乎都有一种外在的简洁.软件开发本身具有的复杂性和不可见性已经使得开发者之间的交流面临着诸多困难,因此,为了能够形成一致的愿景,所使用的软件过程必须具有一种外在的简洁性.整个软件系统应如何开发?应分成哪些阶段?每个阶段要实现什么样的目标?每个阶段与整体之间的关系是什么?诸如此类的问题,需要整个开发团队达成共识.正因如此,我们不难理解具有外在简洁性的 SCRUM 方法之所以能够长期占据各类敏捷方法首位的原因[29].当然,按照 Royce 的说法,简单的软件过程是不能应对复杂软件项目的开发的.因此,仅仅保持外在的简洁是不够的,还需要尽可能详细地规划每个主要开发阶段的具体流程,才能形成一个具有指导意义的可以工作的软件过程.

发现 8. 对软件过程标签敏捷或者非敏捷没有意义.

国内外软件过程社区都习惯于用敏捷或者非敏捷来给某种软件过程或者软件开发方法作标签.同时,热衷于通过各种案例来对比各自优缺点,并试图提出一些方法来选择和定义合适的软件过程.应该说,这种方式鲜有经得起推敲的成功案例:首先,对于敏捷或者非敏捷,事实上并没有公认的标准,因此,敏捷和非敏捷的界限是模糊的,判断某种方法是敏捷或者非敏捷,并没有坚实的科学依据;其次,分属于两个阵营中的典型实践其本质可能与该阵营宣称的理念并不相符,甚至对立.例如:计划驱动往往会被认为是非敏捷阵营的做法,然而,几乎所有的敏捷过程都有一个计划游戏的实践来完成估算和项目计划.这类项目的进度也经常被要求进行跟踪,出现了偏差就需要纠正,确保实际进展与计划一致.很明显,这就是典型的计划驱动做法.再比如:在很多文献中,规范(disciplined)也被认为是敏捷的对立面[23].事实上,当我们观察整个软件过程的发展历史时发现,不追求规范的那段时间恰恰就是软件开发处于极度混乱的时期.尽管后来的形式化方法或者 Cleanroom 方法有矫枉过正之嫌,但后续的一些软件过程和方法对规范的要求其实一直都很高.例如,有些文献把 XP 列为规范要求最高的软件开发方法之一[23].因此,敏捷或者非敏捷这种单一维度区分软件过程特征的方式可能会带来很多概念上的误导和实践中的混乱.应用本文提出的多维视角来理解软件过程特征,进而选择和定制软件过程可能更为合理.

4 总 结

我们梳理并澄清了软件组织与管理方面的若干概念,尽管这些概念早已存在,然而,大量的文献中都采取了混为一谈的方式来处理这些概念,从而导致很多理念上的争议和实践中的踯躅.更加严重的是:这样的混乱使得我们在面临特定软件项目环境中,仍然不清楚如何有效选择、组合和定制软件开发过程,从而更好地实现项目目标,尽管这是软件过程相关研究和实践的基本目的.有鉴于此,我们结合软件开发的四大本质困难在不同历史时期的演变,论述了软件过程的历史演变,从中试图探索事物发展规律,指导实践.

本文的主要贡献如下.

- 首先,梳理并澄清了软件组织与管理方法中的两个基本概念,即软件项目管理和软件过程管理.尽管看起来是实践者本来应该掌握的概念,但在实践中,这两个概念被频繁地曲解,导致很多争论和理解混乱.如果不对上述两个基本概念加以区分,软件的组织与管理无法阐述清楚.事实上,这个问题在现有的相关主题的文献中,是一个相当普遍的问题;
- 其次,我们提出了一种多维视角来刻画软件过程特征的方法.这不仅提供了对软件过程特征更为清晰而准确的描述,同时也为软件过程的组合、定制以及演化改进提供了基础.相比较于用“敏捷过程”“传

统过程”等方式描述软件过程的特征,我们提出的多维视角粒度更细,也打破了上述不同方法阵营之间的壁垒.同时,一些过程选择和组合的基本规则,在多维视角之下也更加直观,也更加容易实现;

- 第三,我们系统梳理了在不同历史时期软件开发的特征和相应的软件过程的演变.发现:随着软件应用的发展,Brooks 总结的软件开发四大本质难题中的复杂性、可变性和一致性的相对重要性会发生变化,从而推动了软件过程和开发方法相应的进行调整.在这个过程中,技术的发展是一个不容忽视的因素;
- 最后,我们总结了若干发现和反思,试图为实践者在软件过程的选择、组合以及定制时提供一些参考.

References:

[1] Wester R, Koster J. The software behind Moore’s law. IEEE Software, 2015,32(2):37−40.

[2] Humphrey WS. The Watts new? Collection: Columns by the SEI’s Watts Humphrey. Carnegie-Mellon University Pittsburgh Pa Software Engineering Institute, 2009. https://resources.sei.cmu.edu/library/

[3] Andrew S, Jennifer G. Applied Software Project Management. O’Reilly Media, 2015.

[4] Team CP. CMMI® for development, Version 1.3, improving processes for developing better products and services. Technical Report, CMU/SEI-2010-TR-033, Software Engineering Institute, Carnegie Mellon University, 2010.

[5] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. Ruan Jian Xue Bao/Journal of Software, 2003,14(4):721−732 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/721.htm

[6] Centers for medicare & medicaid services (CMS) office of information service. Selecting a development approach. In: Proc. of the Selecting A Development Approach. 2008.

[7] Linger RC. Cleanroom process model. IEEE Software, 1994,11(2):50−58.

[8] Beck KL. Extreme Programming Explained—Embrace Change. Addison-Wesley, 1990. 1−190.

[9] Schwaber K. SCRUM Development Process. 1997. 117−134. http://www.doc88.com/p-1438579221661.html

[10] Stapleton J. DSDM: Dynamic systems development method. In: Proc. of the Technology of Object-Oriented Languages and Systems. 1999. 406.

[11] Palmer SR, Felsing JM. A Practical Guide to Feature-Driven Development. Prentice Hall, 2002.

[12] Karlström D, Runeson P. Integrating agile software development into stage-gate managed product development. Empirical Software Engineering, 2006,11(2):203−225.

[13] Paulk MC, Curtis B, Chrissis MB, Weber CV. Capability maturity model for software. In: Proc. of the IEEE Software. 2001. 49−88.

[14] Chrissis MB, Konrad M, Shrum S. CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley Professional, 2003.

[15] Osterweil LJ. Software processes are software too, revisited: An invited talk on the most influental paper of ICSE 9. In: Proc. of the 19th Int’l Conf. on Software Engineering (ICSE). 1997.

[16] Glazer H, Dalton J, Anderson D, Konrad M D, Shrum S. CMMI or agile: Why not embrace both! 2008. [doi: 10.1109/AGILE.2006.30]

[17] Fritzsche M, Patrick K. Agile methods and CMMI: Compatibility or conflict. e-Informatica Software Engineering Journal, 2007, 1(1):9−26.

[18] Gandomani TJ, Zulzalil H. Compatibility of agile software development methods and CMMI. Indian Journal of Science & Technology, 2013,6(8):5089−5094.

[19] Humphrey WS. Managing the Software Process, Vol.1. Reading: Addison-Wesley, 1989.

[20] Sokovic M, Pavletic D, Pipan KK. Quality improvement methodologies: PDCA cycle, RADAR matrix, DMAIC and DFSS. Journal of Achievements in Materials and Manufacturing Engineering, 2010.

[21] McFeeley B. IDEAL: A user’s guide for software process improvement. In: Proc. of the HICSS. 1996.

[22] El Emam K, Melo W, Drouin JN. SPICE: The Theory and Practice of Software Process Improvement and Capability Determination. Wiley-IEEE Computer Society Press, 1997.

[23] Boehm B, Turner R. Balancing Agility and Discipline: A Guide for the Perplexed, Portable Documents. Addison-Wesley Professional, 2003.

[24] Boehm B, Turner R. Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods. In: Proc. of the 26th Int'l Conf. on Software Engineering (ICSE 2004). IEEE, 2004. 718−719.

[25] Kalus G, Kuhrmann M. Criteria for software process tailoring: A systematic review. In: Proc. of the 2013 Int'l Conf. on Software and System Process. ACM Press, 2013. 171−180.

[26] Kuhrmann M, Diebold P, Münch J, Tell P, Garousi V, Felderer M, Trektere K, McCaffery F, Linssen O, Hanser E, Prause CR. Hybrid software and system development in practice: Waterfall, scrum, and beyond. In: Proc. of the 2017 Int'l Conf. on Software and System Process. ACM Press, 2017. 30−39.

[27] Rong G, Boehm B, Kuhrmann M, Tian E, Lian S, Richardson I. Towards context-specific software process selection, tailoring, and composition. In: Proc. of the 2014 Int'l Conf. on Software and System Process. ACM Press, 2014. 183−184.

[28] Fowler M. The new methodology. Wuhan University Journal of Natural Sciences, 2001,6(1-2):12−24.

[29] Anderson DJ. Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press, 2010.

[30] Humphrey WS. PSP: A Self-Improvement Process for Software Engineers. Addison-Wesley Professional, 2005.

[31] Humphrey W, Over J. Introduction to the Team Software Process. Addison-Wesley Professional, 1999.

[32] Turk D, France R, Rumpe B. Limitations of agile software processes. In: Proc. of the Computer Science. 2014. 43−46.

[33] Rong G. Are we ready for software process selection, tailoring, and composition. In: Proc. of the 2014 Int'l Conf. on Software and System Process. ACM Press, 2014. 185−186.

[34] Benedicenti L, Ciancarini P, Cotugno F, Messina A, Sillitti A, Succi G. Improved agile: A customized scrum process for project management in defense and security. In: Mahmood Z, ed. Proc. of the Software Project Management for Distributed Computing, Computer Communications and Networks. Cham: Springer-Verlag, 2017.

[35] David S. Applying the personal software process (PSP) with Ada. In: Proc. of the ACM Sigada Int'l Conf. on Ada ACM. 1998. 219−228.

[36] Yang WR, *et al*. Investigating the benefits of combining PSP with agile software development. In: Proc. of the Int'l Workshop on Evidential Assessment of Software Technologies (EAST 2011), Conjunction with ENASE 2011. Beijing: DBLP, 2011. 36−43.

[37] Park YK, *et al*. A study on the application of six Sigma tools to PSP/TSP for process improvement. In: Proc. of the IEEE/ACIS Int'l Conf. on Computer and Information Science 2006, and 2006 IEEE/ACIS Int'l Workshop on Component-Based Software Engineering, Software Architecture and Reuse. ICIS-Comsar IEEE, 2006. 174−179.

[38] Brooks FP. No silver bullet. IEEE Computer, 1987,20(4):10−19.

[39] Boehm BW. Software and its impact: A quantitative assessment. In: Proc. of the TRW Systems, Engineering and Integration Division. 1973.

[40] Boehm B. A view of 20th and 21st century software engineering. In: Proc. of the 28th Int'l Conf. on Software Engineering. ACM Press, 2006. 12−29.

[41] Naur P, Randell B. Software engineering: Report of a conference sponsored by the NATO science committee. In: Proc. of the Brussels, Scientific Affairs Division'68. Garmisch: NATO, 1969. 231.

[42] Randell B, Buxton JN. Software engineering techniques: Report of a conference sponsored by the NATO science committee. In: Proc. of the Brussels, Scientific Affairs Division, 1969. Rome: NATO, 1970. 164.

[43] Standards Australia. Information Technology: Software Life Cycle Processes. Australian/New Zealand Standard, 1997. https://infostore.saiglobal.com/preview/293342316700.pdf?sku=115590_SAIG_AS_AS_

[44] Hoare CA. An axiomatic basis for computer programming. Communications of the ACM, 1969,12(10):576−580.

[45] Dijkstra E. Cooperating Sequential Processes. Academic Press, 1968.

[46] Royce WW. Managing the development of large software systems: Concepts and techniques. In: Proc. of the 9th Int'l Conf. on Software Engineering. IEEE Computer Society Press, 1987. 328−338.

[47] Coad P. DOD-STD-2167, defense system software development: Point, counterpoint, and revision A. In: Proc. of the Computer Standards Conf., Computer Standards Evolution: Impact and Imperatives. IEEE, 1988. 47−50.

[48] TP026B R. Rational unified process. Cited, 2017. https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

[49] Kroll P, Kruchten P. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison-Wesley Professional, 2003.

[50] Hoch F, Kerr M, Griffith A. Software as a service: Strategic backgrounder. In: Proc. of the Software & Information Industry Association (SIIA). 2001.

[51] Larman C, Basili VR. Iterative and incremental developments, a brief history. IEEE Computer, 2003,36(6):47−56.

[52] Boehm B. A spiral model of software development and enhancement. ACM SIGSOFT Software Engineering Notes, 1986,11(4): 14−24.

[53] Crinnion J. Evolutionary Systems Development, A Practical Guide to the Use of Prototyping within A Structured Systems Methodology. New York: Plenum Press, 1991. 18.

[54] Beck K, *et al*. Manifesto for Agile Software Development. 2001. http://agilemanifesto.org/

[55] Ahmad MO, Markkula J, Oivo M. Kanban in software development: A systematic literature review. In: Proc. of the 39th EUROMICRO Conf. on Software Engineering and Advanced Applications (SEAA). IEEE, 2013. 9−16.

[56] Abrahamsson P, Warsta J, Siponen M, Ronkainen J. New directions on agile methods: A comparative analysis. In: Proc. of the 25th Int'l Conf. on Software Engineering (ICSE 2003). Washington: IEEE Computer Society, 2003. 244−254.

[57] Nerur SP, Mahapatra RK, Mangalaraj G. Challenges of migrating to agile methodologies. Communications of the ACM, 2005,48(5): 72−78.

[58] VersionOne. Annual State of Agile Development Survey. 2018.

[59] Raymond ES. The Cathedral and the Bazaar. O'Reilly Media, 1999.

[60] Stol KJ, Fitzgerald B. Inner source-adopting open source development practices in organizations: A tutorial. IEEE Software, 2015, 32(4):60−67.

[61] Howe J. The rise of crowdsourcing. Wired Magazine, 2006,14(6):1−4.

[62] Bass L, Weber I, Zhu L. DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 2015.

[63] Kim G, Humble J, Debois P, Willis J. The DevOps Handbook: How to Create World-Class Agility. 2016.

[64] Roberts M. Serverless architectures. 2016. 4. https://martinfowler.com/articles/serverless.html

[65] Sbarski P, Kroonenburg S. Serverless Architectures on AWS: With Examples Using AWS Lambda. Manning Publications Company, 2017.

[66] Kim G, Behr K, Spafford K. The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win. It Revolution Press, 2014.

[67] Weinberg GM. The Psychology of Computer Programming. New York: Van Nostrand Reinhold, 1971.

[68] DeMarco T, Lister T. Peopleware: Productive Projects and Teams. Addison-Wesley, 2013.

附中文参考文献:

[5] 梅宏,陈锋,冯耀东,杨杰.ABC:基于体系结构、面向构件的软件开发方法.软件学报,2003,14(4):721−732. http://www.jos.org.cn/1000-9825/14/721.htm

荣国平(1977－),男,江苏苏州人,博士,助理研究员,CCF 专业会员,主要研究领域为软件过程改进,DevOps.

邵栋(1976－),男,副教授,CCF 专业会员,主要研究领域为软件过程,高科技市场理论,敏捷软件开发,软件工程教育.

张贺(1971－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为软件过程,软件体系结构,服务计算,经验软件工程领域的科研和实践.

王青(1964－),女,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为以过程为中心的软件质量管理技术,建模技术,知识管理技术,软件协同工作技术.