软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(9):2642−2664 [doi: 10.13328/j.cnki.jos.006275]

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

领域驱动设计模式的收益与挑战:系统综述*

贾子甲[1,2], 钟陈星[1,2], 周世旗[1,2], 荣国平[1,2], 章 程[3]

[1](南京大学 软件学院,江苏 南京 210023)

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

[3](安徽大学 计算机科学与技术学院,安徽 合肥 230601)

通讯作者: 荣国平, E-mail: ronggp@nju.edu.cn

摘 要: 背景:近年来,领域驱动设计(domain driven design,简称 DDD)作为一种软件设计方法在业界中逐渐流行起来,并形成了若干应用的固有范式,即领域驱动设计模式(domain driven design pattern,简称 DDDP).然而,目前软件开发社区却仍然对 DDDP 在软件项目中的作用缺少较为全面的了解.目的:旨在揭示 DDDP 的应用情况,即哪些 DDDP 被应用到了软件开发中,以及其所带来的收益、挑战及相应的缓解挑战方法.方法:应用系统化文献综述方法,对 2003 年~2019 年 7 月之间发表的相关文献进行了识别、筛选、汇总和分析.结果:通过结合手动检索、自动检索和滚雪球等过程,覆盖了 1 884 篇相关文献,经过筛选,最终得到 26 篇高质量文献,对应 26 个独立的研究.总结了基础研究中 DDDP 的应用概况,即已经被应用到软件开发中的 DDDP 以及应用 DDDP 所获得的 11 项收益、17 个挑战以及相应的缓解挑战方法.结论:因为对领域知识非常重视,领域驱动设计能够帮助实践者更好地进行软件设计,但在具体应用领域驱动设计模式时却存在着诸多挑战.虽然目前存在一些缓解方法能够在一定程度上应对挑战,但是仍然存在很多不足.通过系统文献综述,填补了学术界在这一领域的空白.考虑到 DDDP 的实践价值与当前理论成熟度的不匹配,未来工业界和学术界应该给予该领域更多关注.

关键词: 系统文献综述;经验研究;领域驱动设计;软件设计

中图法分类号: TP311

中文引用格式: 贾子甲,钟陈星,周世旗,荣国平,章程.领域驱动设计模式的收益与挑战:系统综述.软件学报,2021,32(9):2642−2664. http://www.jos.org.cn/1000-9825/6275.htm

英文引用格式: Jia ZJ, Zhong CX, Zhou SQ, Rong GP, Zhang C. Benefits and challenges of domian driven design patterns: Systematic review. Ruan Jian Xue Bao/Journal of Software, 2021,32(9):2642−2664 (in Chinese). http://www.jos.org.cn/1000-9825/6275.htm

Benefits and Challenges of Domian Driven Design Patterns: Systematic Review

JIA Zi-Jia[1,2], ZHONG Chen-Xing[1,2], ZHOU Shi-Qi[1,2], RONG Guo-Ping[1,2], ZHANG Cheng[3]

[1](Software Institute, Nanjing University, Nanjing 210023, China)

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

[3](School of Computer Science and Technology, Anhui University, Hefei 230601, China)

* 基金项目: 国家自然科学基金(62072227, 61802173); 国家重点研发计划(2019YFE0105500); 江苏省政府间双边创新项目(BZ2020017);计算机软件新技术国家重点实验室(南京大学)创新项目(ZZKT2019B01)

Foundation item: National Natural Science Foundation of China (62072227, 61802173); National Key Research and Development Program of China (2019YFE0105500); Intergovernmental Bilateral Innovation Project of Jiangsu Province (BZ2020017); Innovation Project of State Key Laboratory for Novel Software Technology (Nanjing University) (ZZKT2019B01)

本文由“面向持续软件工程的微服务架构技术”专题特约编辑张贺教授、王忠杰教授、陈连平研究员和彭鑫教授推荐.

论文前两位作者对论文工作的贡献相当,作为共同第一作者.

收稿时间: 2020-09-15; 修改时间: 2020-10-26; 采用时间: 2020-12-15; jos 在线出版时间: 2021-01-15

Abstract: In recent years, domain driven design (DDD), as a software design method, has gradually become popular in the industry and formed several inherent paradigms of application, namely domain driven design pattern (DDDP). However, the software development community still lacks a comprehensive understanding of the role of DDDP in software projects. Objective: This study aims to reveal the application status of DDDP, including which DDDP is applied to software development, the benefits, challenges, and mitigation methods for challenges. Methods: In our study, a systematic literature review is performed to identify, screen, summarize and analyze the relevant literature published between 2003 and July 2019. Results: Through the combination of manual retrieval, automatic retrieval and snowballing, this paper covered 1 884 relevant literatures, and after screening, 26 high-quality literatures were finally obtained, corresponding to 26 independent studies. This study summarized theoverview of DDDP in the primarystudies, including the 11 benefits, 17 challenges and the mitigation methods of challenges for the DDDP which applied in software development. Conclusion: DDD can help practitioners design software better since its prominent emphasis on domain knowledge, but there are still some challenges when applying DDD patterns. While these mitigation methods may tacklethe challenges to a certain extent, there are also some deficiencies remained. This study fills in the knowledge gaps in this field through SLR. Considering the mismatch between the practical value of DDDP and the current theoretical maturity, the industry and academia should pay more attention to this field in the future.

Key words: systematic literature review; empirical study; domain-driven design; software design

领域驱动设计(domain-driven design,简称 DDD)是一种在软件设计中应该遵循的思维方式,其目标在于加快具有复杂需求的软件项目的研发速度[1].DDD 中最重要的理念是通过构建领域模型来解决软件开发的复杂性问题[1],因为在大多数软件项目中,最困难的往往是解决业务领域复杂性,而非技术复杂性.DDD 的一个显著特征就是设计与开发的绑定,DDD 强调软件设计概念必须在开发过程中得以成功实现[2].为了实现以上这些愿景,在 DDD 方法中,将一组设计实践、技术和原则合并为标准模式,即所谓的领域驱动设计模式(domain driven design pattern,简称 DDDP)[1],它们则构成了领域驱动设计的主体.

自从 2004 年“领域驱动设计”被首次提出以来[1],这一概念随着软件开发组织的广泛应用逐渐流行起来.一些实践者为 DDD 社区做出了巨大的贡献,例如:Nilsson[3]讲述了如何通过结合 DDD 来构建.NET 应用程序,Vernon[4]对于 DDD 的概念给出了新的解释,Millett 和 Tune[5]阐释了对 DDD 的实践、模式和原则的全面理解.同时,DDD 也被 IBM[6]、阿里巴巴[7]等大型企业采用,以支撑大规模应用.此外,DDD 社区也活跃着多个面向工业界的会议,例如 DDDSummit(https://ddd-summit.de),EXPLORE DDD(http://exploreddd.com)和 DDDConference (http://www.ddd-china.com).

DDD 的立足点是解决软件开发中的复杂性问题,这顺应了当今软件系统复杂度不断提升的趋势;此外,DDD 与当前主流的分布式架构设计风格——微服务架构(micro service architecture)[8]的关系非常紧密[9,10].而近年来,微服务架构已经越来越多地应用在各类大型软件系统当中.上述技术特点和发展趋势,使得 DDD 在业界越来越流行.

然而,领域驱动设计也面临着一些困境.在学术研究方面,相比于在业界实践中的流行程度,DDD 相关的研究关注度明显不足.在业界实践方面,想要成功地在软件开发中实践领域驱动设计,特别是 DDDP,往往具有一定的挑战性.开发人员在应用限界上下文(bounded context)、聚合(aggregate)[4]等比较抽象的 DDDP 时往往感到非常困惑,比如在分布式系统中应用限界上下文模式就存在一定困难[11],这在最近的一项微服务相关访谈研究[12]中得到了证实.

而 DDDP 的实际应用情况如何?其能够为软件开发带来何种收益?又将造成什么挑战?上述问题目前尚未得到较为系统化的回答.因此,为了更加系统地了解 DDDP 的应用现状、收益、挑战及相应的解决方法,本文进行了系统文献综述(systematic literature review,简称 SLR)来调研已发表的基础研究(primary study).系统文献综述是软件工程领域的一种主要的“循证”方法,该方法通过合成当前研究中的高质量证据,以减少单个基础研究的偏差,从而达到辅助软件项目的决策过程的目的.自从这种“循证”方法被软件工程社区采用以来,各领域的许多重要研究都应用了系统文献综述这一方法[13−15].考虑到软件开发组织能够从这种系统文献综述中获取宝贵经验,并且目前的学术界尚未发表有关领域驱动设计的综述性文章,因此,本文的工作对于学术界和产业界均具有一定的借鉴意义.

具体而言,本研究工作的主要贡献包括以下 3 个方面:首先,本文揭示了 DDDP 在软件开发项目中的实际应用情况;其次,总结了应用 DDDP 所带来的收益,便于实践者能够在软件项目中更好地把握机遇;再次,本文对于 DDDP 应用实践中所面临的挑战以及应对挑战的缓解方法进行了论述,能够帮助实践者更好地应对挑战,并为研究者提供了未来可能的探索方向.

本文第 1 节介绍 DDDP 的背景知识.第 2 节介绍本文所采用的系统文献综述方法.第 3 节给出综述的结果.第 4 节对于综述结果进行深入分析和讨论.第 5 节分析本研究中存在的效度威胁.最后,第 6 节给出对于应用 DDDP 的总结与展望.

1 背景介绍

1.1 领域驱动设计的本质

领域模型(domain model)作为设计具有复杂业务规则的企业应用程序的一种软件设计模式,该模式可以被看作是一个具有行为和数据的领域对象模型[16].领域模型正是领域驱动设计的基础.具体而言,DDD 方法说明了如何利用 DDDP 从业务中抽象出领域模型,并将其置于软件开发的核心位置[1].

DDD 具有几个基本原则.

- 第一,在首先提出 DDD 的书籍中[1],Evans 强调设计概念必须在代码中成功实现,否则,它们将会变成抽象的讨论.DDD 通过引入模型驱动设计(model driven design)建模范式及其构造块,弥补了模型与可运行的软件之间的差距;
- 第二,DDD 还提倡迭代开发,并说明了如何借助敏捷[17]中的迭代式领域建模加速软件开发[1].领域驱动设计与具体实现过程间的紧密关系,使得 DDD 比其他软件设计方法更具有实用性;
- 第三,在实践 DDD 的过程中,开发人员和领域专家之间需要展开紧密协作,这是因为 DDD 追求的领域模型需要依靠头脑风暴的创造性和对领域的深入了解[1];
- 第四,DDD 是一种软件设计方法,而任何设计出来的领域模型都应该与架构无关[4].也就是说,除了领域模型,在软件开发过程中仍然需要架构设计,比如微服务架构和六边形架构[4].

1.2 领域驱动设计模式概览

DDD 由 Evans[1]作为一种大型模式语言(pattern language)引入,其由一组相互关联的模式组成.模式语言提供了讨论问题的交流术语,它定义了特定场景、特定问题的解决方案,其主要目的是帮助开发者解决在设计和编程中遇到的通用问题.模式语言在软件工程中被广泛地应用,比如设计模式(design pattern)、企业架构模式(enterprise architecture pattern)等.DDD 与 DDDP 的关系,正如同面向对象设计(object-oriented design,简称 OOD)与面向对象设计模式(即设计模式)的关系.

对于领域驱动设计而言,最基本的模式是通用语言(ubiquitous language),它是一种供不同涉众(如开发人员和领域专家)共同使用的语言,主要用于辅助领域建模[4].一种通用语言只适用于单个限界上下文(bounded context),后者作为一个显式的模型边界来维护模型的完整性.根据 Vernon[4]的观点,除了通用语言之外的 DDDP,要么属于战略设计模式(strategic design pattern),要么属于战术设计模式(tactical design pattern).

战略设计模式旨在应对具有多个领域模型的大型软件开发项目的复杂性,其中,每个领域模型都属于单独的限界上下文.具体而言,限界上下文作为一个显式的模型边界来维护模型的完整性,从而避免了模型之间的相互连接使彼此变得模糊[1].限界上下文以外的其他战略设计模式关注如何管理不同限界上下文(如上下文映射和职责分层)之间的关系,比如,上下文映射(context map)负责描述不同领域模型间的通信,而职责分层(responsibility layer)则站在更高的抽象层级来组织不同领域模型间的概念依赖关系.

战术设计模式负责根据通用语言对单个限界上下文进行领域建模[4],并结合面向对象的原则来绑定领域建模和编码实现.战术建模的典型模式包括实体(entity)、值对象(value object)、聚合(aggregate)、服务(service)、资源库(repository)等.具体来讲:实体和值对象用于对具有不同领域特征的领域对象进行建模;聚合将一组领域

对象绑定为一个整体,以控制事务;服务则充当领域模型的接口,具有无状态的特点;而资源库则用于封装领域对象的数据库访问操作.

2 综述方法

本系统性文献综述根据 Kitchenham 等人的指南[18]开展研究.本节首先提出驱动本工作开展的研究问题,其次描述了数据收集所遵循的详细策略和流程,最后描述了本研究的数据抽取和合成过程.

2.1 研究问题

本研究的总体目标是形成在软件开发中应用领域驱动设计模式的系统性理解,了解应用领域驱动设计模式的实践情况,包括其带来的收益、挑战及应对挑战的缓解方法.为此,本研究提出了如表 1 所示的具体研究问题.其中:研究问题 1 是为了了解 DDDP 在软件开发项目中的应用现状;研究问题 2 是为了调研应用 DDDP 所带来的收益情况;研究问题 3 是为了总结应用 DDDP 所面对的挑战以及应对挑战的缓解方法,并挖掘未来可能的研究方向.

Table 1 Research questions

表 1 研究问题

方面	编号	研究问题
现状	问题 1	哪些 DDDP 被应用于软件开发?
收益	问题 2	应用 DDDP 所带来的收益是什么?
挑战	问题 3	应用 DDDP 需要面对的挑战是什么?

2.2 文献收集

本小节描述了系统文献综述中的数据收集过程,包括了文献检索、文献筛选、文献整合、滚雪球、质量评估和模式识别等.文献收集的总体流程(于 2019 年 7 月进行)如图 1 所示.本文的 3 位作者在来自手动检索、自动检索以及滚雪球过程的 1 884 篇文献中最终确定了 26 篇高质量的基础研究(primary study)作为研究集合.

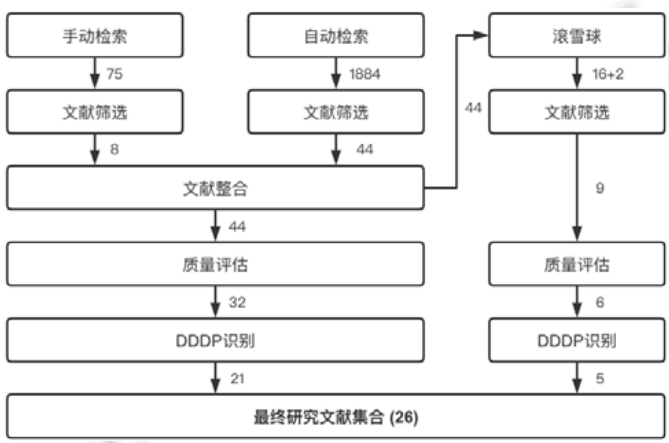


Fig.1 Literaturecollection procedure

图 1 文献收集过程

2.2.1 文献检索

为了尽量降低遗漏任何相关文献的风险,本文采用了多种文献检索策略.此外,根据 Kitchenham 等人的指导方针[18],我们通过试点(pilot)文献综述确定了一组相关文献,以用于验证检索过程的完整性.这组已知文献包括:

- Challenges of domain-driven microservice design: A model-driven perspective [J];
- Towards a UML profile for domain-driven design of microservice architectures [C];
- Designing microservice-based applications by using a domain-driven design approach [J];

- An ontology-based approach for domain-driven design of microservice architectures [J].

接下来对检索过程的一些关键步骤进行详细说明.

(1) 手动检索

为了提供更全面的自动检索字符串,本研究首先进行了手动检索.本过程所选取的两种出版源分别是:

① 已知文献集合的出版源;

② 软件设计和架构相关的期刊与会议.

手动检索的过程首先由本文的两位作者独立进行,之后将检索结果合并到一起,即:只要一位作者认为该文献与本研究相关,就会被纳入此阶段的文献集合中.

表 2 展示了从各个出版源检索到的相关文献数量.

Table 2 Publication venues included in manual search

表 2 手动检索的出版源

出版源	类型	数量
IEEE Software	期刊	26
Int'l Conf. on Software Engineering (ICSE)	会议	16
Object-Oriented Programming, System, Languages & Application (OOPSLA)	会议	8
Information and Software Technology (IST)	期刊	6
Software and System Modeling (SoSyM)	期刊	6
Int'l Conf. on Knowledge and Systems Engineering (KSE)	会议	4
IEEE Trans. on Software Engineering (TSE)	期刊	3
ACM Trans. on Software Engineering and Methodology (TOSEM)	期刊	2
European Joint Conf. on Theory and Practice of Software (ETAPS)	会议	2
IET Software (IETS)	期刊	1
ACM SIGSOFT Symp. on the Foundation of Software Engineering/ European Software Engineering Conf. (FSE/ESEC)	会议	1
Journal of Systems and Software (JSS)	期刊	0
Int'l Conf. on Software Architecture (ICSA)	会议	0
Int'l Conf. on Web Services (ICWS)	会议	0
European Conf. on Software Architecture (ECSA)	会议	0
总数	–	**75**

(2) 自动检索

在这一阶段,本文作者检索了 3 个主要的在线学术数字图书馆:① IEEE Xplore(https://ieeexplore.ieee.org/Xplore);② ACM Digital Library(https://dl.acm.org);和③ ScienceDirect(https://www.sciencedirect.com);以及一个索引系统:④ Scopus(https://www.scopus.com).对于检索字符串,本文作者确定了研究问题中最普遍的关键字(即"DDD"),并决定只包含它和它的相关同义词进行检索,这些同义词是从手动检索结果和已知的论文集合中获得的.通过合并 OR 操作符,本文使用的最终检索字符串如下:

("DDD" OR "domain driven design*" OR "domain-driven design*" OR "domain driven approach*" OR "domain-driven approach*" OR "domain driven develop*" OR "domain-driven develop*")

每个检索源的详细检索结果见表 3.

Table 3 Intermediate result in snowballing

表 3 滚雪球过程的中间结果

在线数字图书馆	文献数量
IEEE Xplore	39
ACM Digital Library	45
ScienceDirect	1 214
Scopus	586
总数(未去重)	1 884

(3) 滚雪球

在手动检索与自动检索之后,本研究中还使用了反向滚雪球(backward snowballing,简称 BSB)和前向滚雪球(forward snowballing,简称 FSB)方法对检索结果进行补充.考虑到滚雪球的初始论文集决定了该过程的效率和完整性,本研究将手动检索和自动检索所得进行文献选择并合并后的结果(见图 1 和表 4)作为滚雪球的初始集合.注意,所有手动检索的结果都被自动检索的结果覆盖.在这个阶段,我们遵循 Wohlin[19]的指导方法,并且反复迭代滚雪球的过程,直到不再有新的文献被发现.

表 4 中,第 1 次迭代和第 2 次迭代最终分别获得 16 项(5+12=17,其中 1 篇文献重复,最终得到 16)和 2 项新研究文献.

Table 4 Intermediate result in snowballing

表 4 滚雪球过程的中间结果

检索阶段	初始文献数量	来自 BSB 的文献数量	来自 FSB 的文献数量	文献总数
迭代 1	44	5	12	16
迭代 2	16	2	0	2

(4) DDDP 的自动检索

为了进一步确保研究集合的完整性,本研究通过基于 DDDP 名称的自动检索来补充现有的检索策略.具体来说,我们在 Scopus 中执行了另一个自动检索过程,其检索的字符串由每个模式的名称以及“domain”(“领域驱动设计”的最通用术语)和“software”(软件工程最通用的术语)组成.比如,与 DDDP“Entity”(及“Entities”)相关的检索字符串为如下:

(“Entit*” AND “domain” AND “software”).

对于 DDDP 的完整列表,我们遵循了 Evans[20]在 2014 年的总结,总共包含 45 个模式.然而,虽然这种检索方法使用 Scopus 中的字段码 TITLE-ABS-KEY 检索到了 7 515 篇论文,但经过文献筛选后,并未获得新的文献.两次检索所得论文数量的巨大反差,可能是因为许多 DDDP 的名称在软件工程中也非常通用,比如服务(service)和资源库(repository)等.因此,本文并没有在图 1 中包含此检索过程.

2.2.2 文献筛选

本系统文献综述使用的文献纳入和排除标准如下所示.注意,只有符合所有纳入标准的基础研究文献才会被纳入,而符合任何一项排除标准的文献都将被排除.

(1) 纳入标准(include criteria)

- IC1:文献提供了关于 DDD 的某种形式的数据.在这一阶段,我们的目标是最大限度地扩大文献范围,以确保研究的完整性;

(2) 排除标准(exclude criteria)

- EC1:发表于 2003 年 DDD 被发表之前的文献(因为 DDD 于 2003 年被提出);
- EC2:无法获得电子版全文的文献;
- EC3:用英语以外的语言撰写的文献;
- EC4:没有经过同行评审的文献;
- EC5:存在更加完整的文献.即同一基础研究(primary study)有多篇文献,此时将最完整的文献纳入.

具体筛选过程如下:

- 前期准备:按主题筛选——我们通过浏览自动检索得到的每篇文献的标题,并确定其是否属于软件工程领域.注意,这个阶段是专门为自动检索设计的,因为一些数字图书馆(例如 Scopus 和 ScienceDirect)不支持基于主题的筛选,或者其基于主题的筛选功能相对有限;
- 第 1 阶段:按标题筛选——我们通过浏览手动检索、自动检索以及滚雪球所得到的文献列表的标题,以确定哪些文献符合纳入/排除标准;
- 第 2 阶段:按关键词和摘要筛选——我们分析通过第 1 阶段的文献的关键词和摘要,进一步确定它们是

否符合选择标准;

- 第 3 阶段:阅读全文筛选——我们通读了通过前两个阶段的文献全文,并保留符合预先定义的筛选标准的文献.

表 5 给出了筛选过程的中间结果.

Table 5 Selection details of literaturefrom different search methods.

表 5 不同检索阶段的文献选择情况

阶段	手动检索	自动检索	滚雪球
准备阶段	–	–	–
阶段 1	75	92	18
阶段 2	28	87	16
阶段 3	8	44	9

在分配选择任务时,我们确保每篇文献的每个选择阶段都由至少两位作者完成.此外,每篇存在争议的文献都要先由第三位作者进行评估,然后再通过讨论达成共识.如果这样还无法解决分歧,则将会和本研究的指导者进行讨论,直到达成共识为止.此外,为了增强我们对文献筛选过程的信心,本研究在自动检索所得文献的筛选过程中引入了 Kappa 评分[21],并且最终 Kappa 评分结果为 0.722(“Good/Substantial”).另外,还有两项证据也可以证明检索和筛选过程的全面性.

- 所有已知集合中的文献(来自试点文献综述)都可以通过检索过程找到,并且被选中;
- 从手动检索中选取的所有文献都包含在自动检索的选择结果中.

2.2.3 质量评估与模式识别

在研究集合的确定过程中,本工作还引入了质量评估(quality assessment)来排除一些质量较低的基础研究.根据 Dyba 等人[22]的质量检查表,本研究制定了质量评估标准,该标准通过了本研究指导者的审查.在质量评估过程中,我们根据评分情况排除了共 15 篇质量得分较低的基础研究.

在经过质量评估之后,本研究进行了模式识别,进一步排除了没有提及任何 DDDP 的基础研究(图 1).之所以将模式识别作为一个环节而不是一项文献筛选标准,是因为一些目标文献可能在另一些文献的滚雪球阶段被纳入进来,而后者可能并没有提及任何 DDDP 而只与领域驱动设计理论相关.这样做的目的是在收集文献时尽可能全面地覆盖相关文献.我们排除了手动检索和自动检索中的 11 篇论文以及滚雪球检索中的 1 篇论文,最终得到了 26 篇高质量基础研究.

2.3 数据抽取与合成

首先给出指导数据抽取过程的数据抽取项,然后描述对数据抽取结果的合成(synthesis)方法.在数据抽取和合成环节中,我们主要使用 Nvivo(https://www.qsrinternational.com/nvivo)和电子表格来记录数据.

2.3.1 数据抽取

我们参照 Zhang 和 Budgen[23]的方法定义了数据抽取项(见表 6),其主要由基础信息、研究背景以及研究问题相关信息这 3 部分组成.其中:基础信息指的是文献的标题、作者以及发表时间等内容;研究背景指的是文献的研究兴趣特征,这些信息可能会影响结果的解释方式;研究问题相关信息指的是文献中与 DDDP 相关的知识,包括回答研究问题所需要的数据项.除了描述每个数据抽取项之外,表 6 还说明了本研究是否为这些数据抽取项提供特定的代码,如果代码为“否”,则该抽取项使用自由文本进行提取.为了进一步确保数据抽取表的可靠性,本文作者与指导者一起对其进行了审查,并且随机选择了 3 篇文献进行了试点研究(pilot study),并根据实验结果对数据抽取项进行了改进.

在进行数据抽取时:首先,我们使用格式化的电子表格来统一数据抽取格式;然后,两位作者逐字逐句地阅读文献并抽取相应的数据.在此过程中,来自研究文献的原始文本被原封不动地记录到数据抽取项中.此外,正如 Cruzes 和 Dyba[24]所推荐的,我们也将表格和图形作为数据抽取的素材粘贴到电子表格中.

Table 6 Data extraction items

表 6 数据抽取项

ID	数据抽取项	是否提供代码	解释说明
	基础信息		
D1	标题	否	文献的标题
D2	作者列表	否	文献作者的列表
D3	发表时间	否	文献正式发表的年月
D4	发表源	否	文献发表的期刊或者会议信息
	研究背景		
D5	研究形式	是	研究主要采用的形式,如案例研究
D6	贡献类型	是	研究所作贡献的类型,如提出解决方案
D7	研究问题	否	指导其工作开展的研究问题
	研究问题相关信息		
D8	DDDP 的应用情况(RQ1)	否	研究中 DDDP 应用实践情况,包括模式列表
D9	所讨论的 DDDP 详细数据(包含 a~g 子数据抽取项)	–	–
	a. 模式名称	是	该特定 DDDP 的名称
	b. 模式特征	否	区别于其他 DDDP 的典型特征
	c. 模式意图	否	该 DDDP 被引入的目的
	d. 范例用法	否	该 DDDP 的使用建议或者范例用法
	e. 收益(RQ2)	否	应用该 DDDP 的实际价值
	f. 挑战(RQ3)	否	应用该 DDDP 所需要面对的问题和挑战
	g. 缓解方法(RQ3)	否	应对相应挑战的技巧或解决方案

2.3.2 数据合成方法

为了深入了解 DDDP,本研究的数据抽取过程产生了大量的定性数据.此外,在 DDDP 相关的基础研究中,同一个词语在不同的上下文中可能具有不同的含义,比如“服务(service)”可能表示一种 DDDP,但也可能表示软件工程中非常普遍的“Web 服务”[25],这为数据合成造成了一定阻碍.因此,本研究需要使用一种合适的定性数据合成方法,来保证能够合成出语义合适的结果.

本研究在进行定性数据合成的过程中,使用了来源于扎根理论(grounded theory)[26]的两个抽象层次的编码方法,即开放码(open codes)和轴向码(axial codes)[27].本研究所使用的编码方法是 Braun 和 Clarke[28]提出的主题分析(thematic analysis),我们利用该方法获得了各种定性数据(见表 6)的聚合结果.

这里简单介绍数据合成过程的一些细节.在熟悉了基础研究文献之后,两位作者分别对提取到的原始文本进行开放编码,即识别文本的语义内容和隐含内容,并将其与原始数据一起记录下来.例如,示例抽取数据的开放编码结果是“更好地理解领域需求”和“更有效的沟通”.当所有的数据集都完成了初始编码后,我们对不同的代码进行分析和比较,并将它们组合成主题.例如,开放编码“更好地理解领域需求”被整理成主题“促进业务理解”.之后,我们进一步对于候选主题进行了细化,比如因为证据不足而放弃了某些开放编码.在这个过程中,NVivo 被用来管理编码和主题之间的层次结构.此外,考虑到这一过程极大地依赖于经验和感知,在执行这一阶段工作之前,所有数据分析和合成的参与者都被要求仔细研读 DDD 相关的主流著作[1,3,4,20,29].

3 研究结果

领域驱动设计模式的应用最终表现在一系列相关活动中,因此,本研究识别出了应用 DDDP 的相关活动,并以此为基础来组织应用 DDDP 所带来的收益与挑战.如图 2 所示,根据基础研究[30],应用 DDDP 的相关活动主要分为 4 类,即领域分析(domain analysis)、领域设计(domain design)、领域模型实现(domain model implementation)和普适性活动(umbrella activity),由此组成了 DDDP 应用框架,并且上述这些活动与 Bruegge 和 Dutoit[31]所描述的传统软件开发活动,即需求获取和分析、设计和实现及测试相对应.

对于 DDDP 应用框架的进一步说明如下.

- 领域分析是指与领域专家一起探索领域知识的过程.经过领域分析,将得到初始的领域模型;
- 领域设计是指将模型分成不同部分(每个部分对应着独立的限界上下文),然后扩展和细化每个限界上

下文的过程,以此为开发实现做准备;

- 领域模型实现是指将模型转换为可运行代码,这一过程还通过检查模型为模型设计提供反馈;
- 普适性活动指的是应用 DDDP 时,可能在领域分析、领域设计、领域模型实现这 3 个阶段都会发生的横切活动,比如构建和更新通用语言.

值得一提的是:在这些活动中,开发团队可能对领域产生更深刻的洞察,并变更之前所做的设计决策.

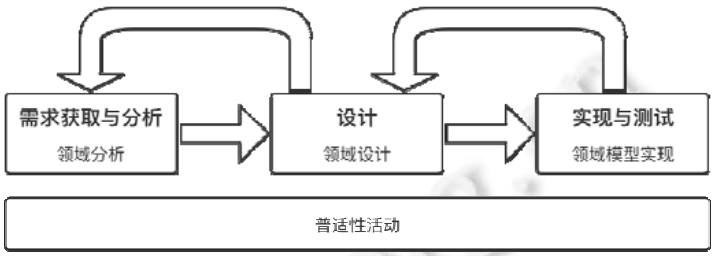


Fig.2 An overview of activities in applying DDD patterns

图 2 应用 DDDP 的活动概览

3.1 研究文献的总体情况

经过收集,本研究总共获得了 26 项基础研究,本部分将简要介绍这些相关研究的总体情况.

- 出版年份

图 3 显示了所收集的文献按出版年份的分布情况.统计数据显示:2006 年之前,社区中没有发表过 DDDP 相关的基础研究文献;而在 2006 年~2015 年期间,文献发表数量一直保持在较低的水平(每年不超过 2 篇).这说明自 2003 年以来,领域驱动设计及其模式在最初的十几年里并没有得到研究者的足够重视.然而 2016 年之后,该领域的论文发表数量逐年增加,这可能与 DDD 和微服务架构在 2016 年的结合有关,特别是在 QCon 伦敦 2016 年大会上[32],《Domain driven design: Tackling complexity in the heart of software》的作者 Eric Evans 提出使用领域驱动设计概念能够减少微服务环境中通用语言(一种 DDDP)的复杂性.本次大会上,Evans 还介绍了 3 种可以帮助管理微服务的 DDD 工具,并建议将每个微服务设计成一个限界上下文(一种 DDDP).DDD 与微服务结合,使得该领域的研究与应用变得更加活跃,这可能是 2016 年后相关文献数量持续增长的一个重要原因.

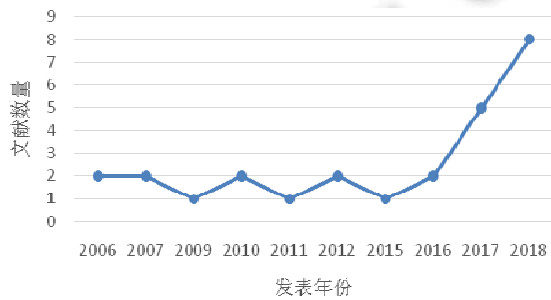


Fig.3 Distribution of literatures by years

图 3 文献发表年份分布

- 作者来源

图 4 显示了所收集的文献按作者来源的分布情况.根据对于论文署名作者及其所属机构的统计数据显示:在本工作所选取的 26 篇基础研究文献中,69.2%(18/26)的文献的全部作者均来自于学术界(包括高等院校以及科研机构等);但是与此同时,也有 26.9%(7/26)的文献作者均来自于业界;此外,还有 1 篇文献由来自于学术界和业界的多位作者共同完成.综上所述,在本文所选取的 DDDP 相关基础研究中,大约 30.8%的研究工作有业界的参与,这也从侧面印证了 DDD 在软件开发业界的流行程度.

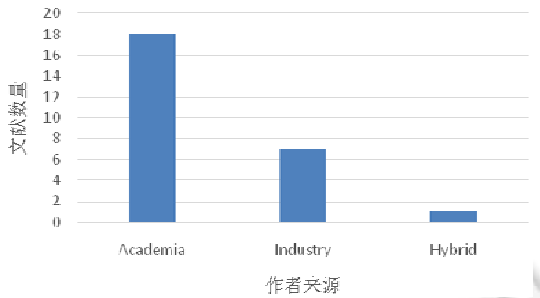


Fig.4 Distribution of literatures by authorship

图 4 文献作者来源分布

- 研究形式

图 5 显示了不同研究形式的分布情况.其中,61.5%(16/26)的文献属于案例研究(case study),当研究对象之间的联系复杂且重要时[18],案例研究是一种非常合适的研究形式.同时,在所有相关研究中,实验都不是主要方法.这可能是由于根据 Easterbrook 等人的理论[33],控制实验不适合真正复杂的软件项目.此外,23.0%(6/26)的文献采用了经验报告(experience report)的形式,这种形式能够帮助读者从中获得实际经验,因此也非常受欢迎.

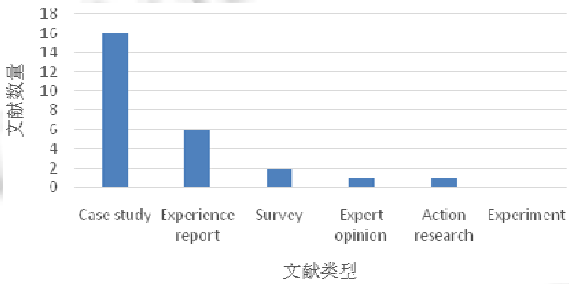


Fig.5 Distribution of literatures bystudy form

图 5 文献研究类型分布

- 贡献类型

如图 6 所示,大多数研究(18/26,69.2%)提出了 DDD 相关的解决方案.这些解决方案研究的目的可以分为两类,其中,77.8%(14/18)的文献致力于利用 DDD 解决具体软件系统的开发问题,22.2%(4/18)文献试图解决 DDD 应用的局限或挑战.此外,15.4%(4/26)的文献论述了将 DDD 应用于实践中所获得的经验和教训.

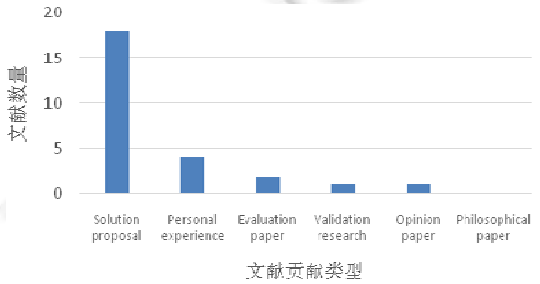


Fig.6 Distribution of literaturesbycontribution type

图 6 文献贡献类型分布

3.2 DDDP的应用情况(RQ1)

表 7 总结了在基础研究集合中,出现频次到达 3 次以上的 DDDP,以及对应的描述和提及这些模式的研究文献.显然,这些模式出现的频次并不平衡.其中,战术设计模式被提及的频次明显高于战略设计,这与 Millett 和

Tune[5]的发现基本一致,即,开发人员更容易注意到领域驱动设计的战术设计模式.总体而言,目前只有 31.1% (14/45)的 DDDP 在基础研究中得到了探讨,这也表明了当前学术界对 DDDP 的研究存在不足.

Table 7 Distribution of DDD patterns in literatures

表 7 研究文献中 DDDP 的分布情况

DDDP	描述	相关文献	计数
通用语言			
通用语言 (ubiquitous language)	一种围绕领域模型构建的共享语言, 主要用于促进团队协作工作中的沟通.	[30],[34–47]	15
战略设计			
限界上下文 (bounded context)	复杂的软件项目中往往存在多个领域模型,而限界上下文通过显式的模型边界来避免与其他模型产生模糊.	[30],[34],[39],[41,42], [44], [47–51]	11
上下文映射 (context map)	软件项目中涉及的限界上下文及其之间关系的表示.	[30],[41],[52,53]	4
分层架构 (layered architecture)	将复杂的软件系统划分成多层,并将领域模型与其他关注点分离开来.	[30],[35],[38],[46],[52],[54],	6
职责分层 (responsibility layer)	用于将模型重构为层次结构,其中每个领域对象的职责与其层次完全对应.	[39],[52,53]	3
防腐层 (anticorruption layer)	封装了两个限界上下文之间的信息转换的隔离层.	[30],[49],[52,53],[55]	5
核心域 (core domain)	使应用区别于其他应用,并且也是领域模型中最值得开发的部分.	[30],[35],[39],[52,53]	5
战术设计			
实体 (entity)	用于建模在整个生命周期中唯一且可以持续变化的领域对象.	[30],[34–37],[39],[41], [43],[44],[47,48],[54],[56]	13
值对象 (value object)	用于建模某些不具有唯一标识的特征的领域对象.	[30],[34–37],[39],[41], [43,44],[47,48],[56]	12
聚合 (aggregate)	用于将一组实体和值对象封装起来,使其作为一个整体进行处理.	[30],[34–37],[39],[41], [42],[44],[47,48],[56,57]	13
资源库 (repository)	为领域对象封装数据库访问的概念框架.	[34–37],[39],[41],[44], [48],[52],[55–57]	12
服务 (service)	用于对充当模型接口的操作建模,并且不封装状态信息.	[30],[34],[36,37],[39], [43,44],[47,48],[56]	10
工厂(factory)	用于封装领域对象的复杂创建逻辑.	[30],[34],[37],[39]	4
领域事件 (domain event)	用于将领域中的活动信息建模为一系列事件.	[30],[34],[58]	3

作为一种模式语言,DDD 由一组相互关联的模式组成(即 DDDP).Vernon[4]指出:如果在实践中忽略通用语言和 DDD 战略设计,而仅仅使用部分战术设计模式,就可能导致领域概念之间的紧密耦合,这样的问题被称为“DDD-Lite 陷阱”.本研究发现:在研究集合中,仅有文献[30,41,42]明确地应用了通用语言、战略设计模式和战术设计模式.有 2 项研究可能落入了 DDD-Lite 陷阱,因为它们只展示了战术模型,而没有提出任何与战略层面实践相关的观点.比如文献[35]在文章中论述了他们如何演进通用语言、战术领域模型以及持久化策略等实现细节,而并没提及战略设计.与此同时,5 篇研究文献在研究中只采用了战略设计,而没有关注战术设计.具体而言,这些研究专注于企业架构层面,主要分析如何利用战略模式(如限界上下文和上下文映射等)来组织具有多个领域模型的大规模结构.此外,基础研究[40]较为特别,它既没有论述战略设计,也没有论述战术设计,而是分析了如何通过映射具有相似语义的领域概念,将两种不同的战术领域模型集成到一起.

3.3 应用DDDP所带来的收益(RQ2)

为了便于读者理解,本文以表格形式展现了将 DDDP 应用到 DDD 活动中的收益与挑战.表 8 展示了应用 DDDP 所带来的收益在领域分析、领域模型实现以及普适性活动中的体现情况,并展示了相关模式和研究文献的证据覆盖范围.除领域分析之外,本研究在 DDD 活动的各个阶段均发现了应用 DDDP 所带来的收益.

Table 8 Benefits of applying DDD Patterns

表 8 应用 DDDP 所带来的收益

阶段	ID	描述	计数	相关模式及研究文献
领域设计	B1	领域间清晰的依赖关系	5	CM[52,53],RL[41,52,53]
领域模型实现	B2	不同实现间轻松切换	3	Repository[37,39],Factory[39]
	B3	减少集成开销	1	AL[55]
	B4	促进接口复用	1	OHS[52]
	B5	明确的代码意图	1	MultiPattern(UL,VO,Entity,Repository,Aggregate)[37]
普适性活动	B6	提升架构质量	4	BC[30],CM[52],LA[54],CD[30]
	B7	更加有效的沟通	4	UL[35,36,38,44]
	B8	促进业务理解	3	CD[52],UL[36],MultiPattern(UL,BC)[42]
	B9	以敏捷方式实现复杂架构的不断演进	2	LA[54],MultiPattern(UL,BC,Aggregate)[42]
	B10	提升开发资源的利用率	1	CD[52]
	B11	应对具有巨大领域复杂性的软件系统	1	MultiPattern(Entity,VO,Service,Aggregate)[56]

注:表格使用简写表示相关模式,其中,CM=“上下文映射”,RL=“职责分层”,OHS=“开放主机服务”,AL=“防腐层”,BC=“限界上下文”,LA=“分层架构”,CD=“核心域”,UL=“通用语言”,VO=“值对象”.MultiPattern 代表这种收益可能与多种 DDDP 的综合作用相关

值得注意的是:一些研究文献所提到的显著收益可能来自于其对于多种 DDDP 的综合作用,而不是某一种 DDDP 的作用.接下来,本文将分阶段讨论应用 DDDP 所带来收益的细节.

3.3.1 领域设计

在领域设计中,应用 DDDP 的收益在于能够使各个领域之间的依赖关系更加明确(B1).上下文映射(context map)和职责分层(responsibility layer)用于组织系统的不同部分:前者表示不同限界上下文之间的关系,每个上下文表示一个特定的领域;后者则根据领域对象的职责,将它们组织成具有清晰依赖关系的层次结构.借助这些 DDDP,我们能够清晰地认识领域之间的关系和依赖.本质上,正如基础研究[53]所宣称的,我们可以利用这些模式了解功能之间的依赖关系.因此在确定系统边界时,不同领域的战术设计知识以及领域之间的关联关系将变得更加明确.理清领域之间的依赖关系还可以帮助开发人员更加深入地了解系统,降低认知复杂性[52],有助于分析系统架构.

3.3.2 领域模型实现

应用 DDDP 有助于领域模型的落地实现.比如应用资源库(repository)和工厂(factory)模式分别可以将存储和初始化的复杂逻辑从领域模型中剥离出来[37,39],因此能够改善软件开发的效率,使得开发人员能够在存储和初始化的不同实现方案之间的更加轻松地进行切换(B2).一方面,借助资源库模式,当需要更改系统时,往往只需要更改相关的代码实现,而不必去检查整个对象模型[37];另一方面,工厂模式隐藏了复杂的初始化逻辑,因此可以在不引用具体实现的情况下轻松地切换实现方式[39].

当涉及到限界上下文之间的通信时,防腐层(anticorruption layer)封装了两个上下文之间领域概念的转换,从而避免一个上下文过多地掌握另一上下文的知识.正如基础研究[55]所指出的:通过将领域模型从执行与其他系统相关的任务中解放出来,防腐层允许实践者在不改变领域模型的情况下集成外部系统,从而减少系统集成的开销(B3),这对于需要与遗留系统或第三方系统进行集成的应用程序至关重要.此外,开放主机服务(open-host service)通过一组服务提供了对限界上下文的访问[1],即,将这些服务发布为设计构件.这样一来,接口的重用性(B4)和松耦合得到了极大改善.

让代码意图更明确(B5),也是应用 DDDP 带来的一个显著收益.DDD 强调对于模型和实现的绑定,所以在编写代码时需要使用与领域建模过程相同的术语[37],也就是通用语言.这样一来,由于整个团队将通用语言作为相互沟通的基础,能够使代码的业务逻辑将变得更加清晰[37].与此同时,借助于通用语言这种模式,不同涉众之间可以直接基于代码而不是海量的文档进行信息交换,这使得团队的沟通、协作效率大大提升.

3.3.3 普适性活动

应用 DDDP 可以提升软件架构的质量属性(B6),如可维护性、可扩展性、可重用性和可测试性等.这一观点在现有研究文献中被广泛认可.首先,限界上下文和上下文映射将一个复杂的领域分解为几个部分,帮助系统

实现模块化.文献[52]的经验表明:实现上下文映射可以帮助开发人员对系统产生更深刻的认识,从而进一步改进系统架构;其次,识别和关注核心域(core domain)可以提高资源利用率,从而以更高效的方式改善软件架构;再次,分层架构(layered architecture)能够将领域模型与其他关注点分离开来,有助于探索领域知识,也有助于确保各层之间的高内聚和低耦合[54].

由于 DDDP 中的通用语言能够作为团队内部的共享术语来降低沟通中的噪声,所以在设计领域模型和分析代码的过程中,通用语言能够为不同的涉众提供高效的通信方式(B7).正是因此,领域专家的参与度得以大大提升,使得更多高价值的领域业务见解在团队内部分享和传递.更好地理解业务(B8)能够使得软件与其所在的领域保持一致,这也正是领域驱动设计的基本观点[4,29].另一方面,对于领域模型特别是核心域的关注,使开发人员能够更好地理解正在开发的软件和其未来愿景,有助于架构决策的制定,如开发资源的高效利用(B10)和系统级优化.

最后,应用 DDDP 能够帮助复杂软件架构以更加敏捷的方式实现演进(B9),并且能够帮助团队应对具有较大领域复杂性的软件系统开发难题(B11).

3.4 应用DDDP所面对的挑战(RQ3)

总结现有研究中对于应用 DDDP 所带来的挑战,能够为学术界提供一些未来的研究方向.然而对于实践者而言,其往往更关心应对挑战的策略和方法.尽管我们难以在现有研究中找到应对 DDDP 所带来挑战的“银弹”,但研究文献中所提出的一些方法,却能够帮助实践者在一定程度上缓解应对挑战的难度,我们将其称为挑战的缓解方法.在本研究中,我们首先在基础研究中提取了应用 DDDP 的 17 个挑战,之后,3 位研究者根据所得到的挑战列表和系统文献综述过程所收集的证据,独立抽取并合成应对挑战的缓解方法,经过分析、讨论与整合,最终得到了 17 个应对挑战的缓解方法.

表 9 列出了在领域分析、领域设计、领域模型实现和普适性活动这些活动中,应用 DDDP 可能面临的挑战以及相应的缓解方法.接下来,本文将分阶段对其进行详细介绍.

Table 9 Challenges of applying DDD Patterns

表 9 应用 DDDP 所需要面对的挑战

ID	描述	计数	相关模式及研究文献	缓解方法
			领域分析	
C1	难以保证领域专家参与获取领域知识过程	1	MultiPattern (UL,BC,CD)[39]	/
			领域设计	
C2	难以确定限界上下文的粒度	1	BC[51]	M1:考虑组织结构[30,44] M2:确定职责的边界[41] M3:通过分析用例的工作流聚类相关功能[49] M4:利用事件风暴探索领域[42] M5:比较不同团队的领域模型[42]
C3	难以对领域进行优先级排序并确定核心领域	1	CD[52]	/
C4	难以确定聚合的粒度	1	Aggregate[58]	/
C5	领域层之外的其他层缺少指导规范	1	LA[30]	M6:使用其他工件来补充遗漏的规范[30]
C6	缺少对模型形式化表达的支持	2	MultiPattern(UL,BC)[44], MultiPattern(UL,BC)[47]	M7:为建模定义 UML 配置文件[44] M8:用于表达领域模型及其关系的语义的元模型[47]
C7	领域专家难以理解建模所使用的技术	1	UL[46]	/

Table 9 Challenges of applying DDD patterns (Continued)

表 9 应用 DDDP 所需要面对的挑战(续)

ID	描述	计数	相关模式及研究文献	缓解方法
领域设计				
C8	缺少合适的建模工具	1	LA[46]	M9:实现了用于建模的 UML 配置[44] M10:Elihu 无需对基础结构进行建模即可实现业务功能建模[56] M11:支持图形建模的 AjiL[48]
C9	领域复杂性使领域模型难以理解	1	MultiPattern(UL,BC,CM,CD)[30]	M12:使用领域视图将模型分解为几个视图[30]
领域模型实现				
C10	实现防腐层是不仅成本高,而且复杂	2	AL[41,53]	/
C11	为服务的实现留下了大量复杂性	1	Service[43]	/
C12	缺少对后续实现的规范	3	BC[48],Service[44],LA[46]	M13:使用建模约定来解决领域模型缺乏规范的问题[48]
C13	领域模型未覆盖基础组件的需求	1	Service[48]	M14:使用中间模型表示技术特性[48]
C14	缺少对模型到代码的自动化转换的支持	1	MultiPattern(BC,CM)[44]	/
普适性活动				
C15	不同的涉众很难对领域概念达成共识	1	UL[37]	M15:专注于信息架构定义一致的术语[37]
C16	不同团队之间的领域模型访问和变更管理困难	1	MultiPattern(Service,BC)[48]	M16:明确模型的使用权限[48]
C17	缺少应用 DDD 的软件开发过程的系统描述	1	MultiPatter(LA,Service)[30]	M17:基于领域驱动设计的软件开发活动概述[30]

注:表格使用简写表示相关模式,其中 CM="上下文映射",RL="职责分层",OHS="开放主机服务",AL="防腐层",BC="限界上下文",LA="分层架构",CD="核心域",UL="通用语言".MultiPattern 代表这种挑战与多种 DDDP 的综合作用相关

3.4.1 领域分析

在领域分析部分,本文发现了一种应用 DDDP 的挑战(C1),但是并没有在研究文献中找到能够缓解该挑战的方法,因此在本部分的缓解方法中,本文根据经验总结出了两条应对建议.

- 挑战

领域驱动设计及模式致力于帮助开发组织深入理解软件的业务领域,以便在系统开发中充分发挥创造力[39].为了更加深入地了解相关领域,无论是形成统一语言还是确定限界上下文和核心域,都强调领域专家与开发人员之间的紧密协作.然而,让领域专家参与领域驱动设计实践并非易事(C1),这成为了阻碍 DDDP 成功应用的一个主要障碍,特别是在所开发的软件系统非常复杂,需要不同领域的多个领域专家共同参与的情况下.Vernon[4]也认同了这一观点,他认为:让领域专家参与软件开发过程,一直以来都十分困难.

- 缓解方法

在本研究所搜集的基础研究文献中,并没有针对领域分析阶段所遇到的 C1 提出任何解决方案.显然,C1 更像是一个组织问题,而不是技术问题.结合 Vernon[4]的观点,我们为应对 C1 提供了两条建议:① 如果软件开发组织决定应用领域驱动设计,那么就应该充分认识领域专家的重要性;② 领域专家指的不是一个职位,而指的是非常了解业务线的人,因此,领域专家可能是销售人员,也可能是产品设计师[4].

3.4.2 领域设计

在领域设计部分,本文发现了 8 种应用 DDDP 所带来的挑战(即 C2~C9),并且对于其中除了 C3,C4 和 C7 之外的 5 项挑战都发现了一些对应的缓解方法(M1~M12).

- 挑战

通过引入限界上下文模式,可以将复杂软件系统拆分为不同部分,拆分后的每个部分都可以独立建立具有清晰边界的领域模型.正如 Newman[8]所建议的:借助限界上下文将相关业务功能组合为业务能力,以此来确定

微服务粒度.这一方法已被软件开发社区广泛接受[51].然而对于限界上下文本身的粒度,DDD 并没有提供任何有助于实现高内聚、低耦合(C2)的指导方法.近期的一项相关研究[59]证实了这一挑战,该工作宣称:使用限界上下文对软件系统进行拆分是非常困难的,因为不同业务功能之间的界限通常并不明显.

将一个领域拆分为多个子域之后,应首先识别出系统中最具决定性的子域,即核心域,并侧重对核心域的投入,以实现收益最大化[4].但在实际中,对开发人员而言,识别系统的核心域往往非常困难(C3),尤其是在开发大型系统时.此外,基础研究[52]还发现,在实践中可能会同时存在多个核心域,因此进一步加剧了这一挑战.

在每个限界上下文内应用战术设计模式主要面临两个挑战:其一,确定聚合的粒度非常困难(C4),这需要在执行单个事务的成本和强一致性两者之间取得平衡[58];其二,使用分层架构(layered architecture)将领域模型与其他模块进行对应时,DDDP 中并未对领域层之外的其他层,如用户界面层、应用程序层和基础设施层等,提供任何相关规范(C5)[1].本质上讲,领域驱动设计强调使用领域模型来展示领域的功能视图,但并没有考虑其他方面[30].然而,这种对领域层之外规范的缺失,将会为实践者带来一定的困扰.

在模型中表达领域知识时,Evans[1]强调可以利用各种媒介进行沟通(不仅是 UML[31])来最大化模型表达的效率.但这就带来了另一项挑战,即,领域驱动设计缺乏对模型的形式化表达的支持(C6).文献[44]指出:DDD 的定义不仅缺少形式化基础,还在某些情况下具有不同的表达方式.在应用微服务架构构建应用程序时,领域模型将分散在不同团队中,因此具有相同语义的领域对象可能会在不同团队中独立演变,从而变得完全不同.这样一来,很可能使得不同团队对相同的领域概念产生不同的理解[47].

在基础研究文献中还提到了其他的一些挑战,如基础研究[46]所描述的,领域模型的建模过程需要领域专家的参与,但却无法保证领域专家能够理解建模过程中使用的全部技术(C7),这对于通用语言的实践造成了一定困难.此外,当前的建模工具还不能完全适用于领域驱动设计(C8).最后一点,虽然 DDD 提供了多种模式,但是领域模型的复杂度依然会影响模型本身的可理解性(C9).

- 缓解方法

为了确定限界上下文的合理粒度(C2),多个研究文献给出了相关建议.首先,文献[30,44]认为,应该考虑团队的组织结构.其次,文献[41]建议找到职责的边界,并将其作为该领域的业务功能.文献[49]建议通过分析用例的工作流将相关功能聚集在一起,并将其划分为不同的领域,也就是限界上下文;此外,通过将领域专家和开发人员召集在一起来讨论业务领域的事件风暴[60],也是一种行之有效的重要手段.通常而言,限界上下文往往需要经过多次迭代才能够最终确定,根据文献[42]的经验,比较来自于不同团队的领域模型,有利于使领域边界更清晰.

为了应对微服务场景中的 C5,也就是领域驱动设计在其他层中没有提供任何规范这一挑战,需要添加其他工件来弥补规范的缺失[30].根据 Fairbanks[61]的建议,在决定向系统中增加新的工件时,需要仔细考虑项目中存在的风险.例如:当使用 DDD 构建基于微服务的 Web 应用程序时,文献[30]根据用户体验设计的指导原则来定义用户界面和交互,根据行为驱动开发(behavior-driven development)来确定应用层的需求,并使用 Spring 框架搭建基础设施层.

为了使建模语言更加形式化(C6),基于在领域驱动设计中应用 UML 元素情况的调研,文献[44]定义了一种用于建模的 UML 配置文件(profile),并充分考虑到了各元素的语法和语义.应用这种 UML 配置文件的案例表明,其能够将基于这种配置建模的领域模型映射到微服务中.文献[48]的研究工作进一步验证了这种 UML 配置文件的有效性.除此之外,本体论是一种在特定领域中描述语义的方法,其能够对系统结构进行形式化建模,比如实体和关系等[62].基于本体论,基础研究[47]提出了一种表达领域模型语义及关系的元模型,以此来确保整个团队对分布式领域驱动微服务设计中的相关领域概念具有相同的理解.

对于建模工具(C8)而言,文献[44]在 Eclipse 和 Papyrus 建模环境的基础上实现了 UML 配置文件;文献[56]在 Eclipse 平台上开发了 Elihu 项目,能够对于领域对象相关的应用功能进行建模,而无需考虑基础设施方面的问题.此外,文献[48]建议使用另一个基于 Eclipse 的工具——AjiL 来实现微服务的图形化建模.AjiL 能够通过建模得到初步的领域模型,并通过代码生成器将模型转化为微服务的具体实现.

对于模型复杂度(C9)而言,概念架构视图[9]被引入并用于将模型拆分为多个领域视图.这样一来,建模者能

够从不同视图进行领域建模,而其中每个视图都代表了特定涉众的观点.

3.4.3 领域模型实现

在领域模型实现部分,本文发现了 5 种应用 DDDP 的挑战(即 C10~C14),并且对于其中的 C12 和 C13 寻找到了对应的缓解方法(M13,M14).

- 挑战

本研究发现:在领域模型的实现过程中,防腐层(anticorruption layer)与服务(service)的实现复杂度较高(对应 C10 与 C11).对前者而言,文献[41]提到,实现防腐层的最大挑战在于控制转换复杂度,也就是开发人员必须充分理解两个限界上下文及其关联;而对后者而言,文献[43]指出,应确定服务的粒度来确保能够提供合理的对相关领域的访问.由于 DDD 主要关注领域模型,并没有对服务实现给出详细指导,因此为实现过程造成了一定的困难.

此外,由于 DDD 侧重于业务领域而非技术领域,这也带来了两项挑战.

- 第一,DDD 中的领域模型没有指定后续的实现规范(C12).举例而言,领域对象的属性可能是无类型的,或者缺少具体的行为方法[48].但是这类信息对于后续实现而言往往是必要的,而由于这类信息的缺失,使得从模型清晰地推断出技术特性变得非常困难;
- 第二,领域模型并不包含基础组件或部署视图中的基础设施(C13),但这类视图或组件信息有时会在很大程度上影响软件系统,特别是在微服务场景中.因此,这类信息需要使用文档加以记录来作为对模型的补充.

模型驱动开发(model-driven development)能够提高领域模型到代码转化的效率,在 DDD 中得到了较广泛的应用.然而,即使在将领域模型正确转换为代码之后,由于领域建模过程的不断迭代,仍然有可能需要重复这一过程.因此,模型到代码转换的自动化(C14)十分必要.而为了使得模型与代码始终保持一致,则需要在建模的每个迭代过程中持续地关注各个领域元素[1].尽管在 DDD 中引入了模型驱动设计,但如何在模型的细化过程中保证代码和模型的一致性[44],仍然是一个难题.

- 缓解方法

建模规约(convention)[48]意味着为建模过程设置需要遵守的规则.建模规约通过为领域模型赋予更多形式化内容,来应对 DDD 中领域模型缺乏实现规范的挑战(C12).文献[48]中提供了一些规则示例,比如在建模时禁止双向关联、事先规定引入微服务接口的 DDDP 等.针对领域模型中缺少基础组件的情况(C13),文献[48]建议利用来自模型驱动开发[63]的中间模型来表示技术特性,如接口模型和部署模型等.前者用于对服务接口进行建模,后者用于对可部署工件进行建模.这些中间模型能够展示特定角度的技术特性,有助于接口类型和部署技术的进一步实现.

3.4.4 普适性活动

在普适性活动部分,本文发现了 3 种应用 DDDP 的挑战(即 C15~C17),并且对于其中的每个挑战寻找到了对应的缓解方法(M15~M17).

- 挑战

文献[37]提到:在形成通用语言的过程中,由于不同涉众使用的语言不同,往往难以对于同一术语形成统一认识(C15).这使得团队在沟通过程中可能会产生误解,想要定义通用语言往往需要付出更多的努力.

领域模型的获取与变更管理,是应用 DDDP 的另一项挑战.Vernon[4]建议由单个团队来开发和维护一个限界上下文及其对应的模型.但是,管理不同团队对领域模型的权限也会遇到挑战(C16),因此必须确定团队对领域模型的权限范围.此外,是否允许团队更改其他团队的领域模型也很重要.以上决策将直接影响到模型的变更情况[38].比如:如果允许团队更改其他团队的限界上下文,服务之间的依赖会变得更加复杂.因此,应该权衡每个决策所带来的损失和收益,从而做出最终决策.

最后,C17 与应用 DDDP 进行软件开发的过程相关.领域驱动设计的核心原则是,使应用程序与领域模型保持一致.正是这一特点,使得领域驱动设计区别于其他传统的软件开发方法[1].然而,虽然这一方法目前已经被广

泛接受,但是其对于如何在系统性软件开发过程应用 DDDP 并没有提供详细描述[30].这使得应用领域驱动设计的活动变得模糊不清,成为实践者们面临的主要挑战,从而影响了适用性.

- 缓解方法

解决 C15[37]的一种途径在于聚焦于信息架构[64],并在沟通中使用统一定义的术语(通用语言).为了提供对模型访问和变更管理的支持(C16),文献[48]论述了在分布式微服务开发背景下架构决策的优缺点,包括完全模型访问和部分模型访问,并提倡采用明确的策略来规范模型管理.为了应对 C17,文献[30]初步确定了 DDD 开发活动集,并与需求确定和分析、设计和实现以及测试[31]等传统软件开发活动保持一致.

4 讨 论

本节综合了基础研究文献中关于应用领域驱动设计模式的现有证据,对于应用DDDP所面对的3个问题进行了讨论,包括应选择战术设计还是战略设计、存在哪些机遇以及如何应对挑战.

4.1 战略设计与战术设计

在 DDDP 中,从战略设计和战术设计角度分别提供了一系列方法.战略设计更关注业务的宏观战略方向[4],而战术设计则聚焦于特定领域模型.本研究发现:相比于战术设计,在 DDD 社区中,实践者更趋向于应用战略设计(18.2%VS.45.5%).这可能是由于:对于企业而言,软件系统的战略设计决策更具有影响力.举例而言:战略设计能够降低分析软件架构时的认知复杂度,并能够促进系统的组件化.

需要强调的是:实践者不应只关注战术设计,而忽略战略设计.过分专注于技术层面可能会导致领域对象之间的细微差异被忽略,从而导致模型的错误融合,甚至会使系统变成大泥球(big ball of mud)[4].Millett 和 Tune[5]认为,这正是许多应用 DDD 的项目失败的原因.因此,本文建议实践者应该同时兼顾战略设计和战术设计,同时,根据自身业务场景来权衡战略性设计和战术设计的使用.

4.2 存在的机遇

现有经验证据(empirical evidence)证明了 DDDP 在软件开发实践中的实际应用价值.然而,在目前的基础研究文献中,仅有 10 种 DDDP 被明确指出对软件开发有所帮助,而其他模式的实际价值还缺少相关证据的支持.

此外,诸如 CQRS(command query responsibility segregation,命令查询的责任分离)[65]等 DDDP 的新兴模式经过社区的发展已经日趋成熟,但由于这些模式并未包含在 Evans 在 2014 年所提出模式列表中,因此学术界目前还缺少相应的研究工作.本文关注的基础研究文献中也缺少对这些模式的相关证据.

需要注意的是:虽然 DDDP 是领域驱动设计标准化设计的重要内容[1],但对于 DDD 而言,深入理解领域知识、迭代建模、统一模型与实现等整体愿景更为关键[1].为了实现以上愿景往往需要综合应用多种 DDDP,同时也需要保持团队组织形式与软件设计理念的一致性.

虽然应用 DDDP 可以为软件开发带来良好的效果,但从现有研究来看,它对其他方面的帮助并不明显.一方面,DDD 强调开发人员和领域专家及其他涉众的协作,那么应用 DDDP 来改善团队组织结构似乎是合理的,比如上下文映射能够明确地展示不同组织之间的动态关系,因此它可能适用于组织大型软件项目的开发工作;另一方面,Vernon[4]认为,应用 DDD 可能会使软件过程的治理问题暴露出来.总的来说,现有研究缺乏对 DDDP 的这些方面的讨论,DDDP 在这些方面的具体作用仍需要更多的实证研究来验证.

4.3 应对挑战

正如 Vernon[4]所说的,使用 DDD 时仍需要面对诸多挑战.在本研究的统计数据中,与挑战相关的数据的来源研究文献比较集中,这表明在目前的学术界研究中,还没有对于应用 DDDP 所面对的挑战达成一致共识.这可能是因为现有的大多数研究工作所关注的主题都比较分散.我们推测:随着 DDD 研究的不断深入,社区将对于应用 DDDP 所面临的挑战逐渐会达成共识.

经过聚合分析[28],可以将本文所报告的应用 DDDP 所面对的挑战划分整理成 4 种类型,见表 10.

- 建模本身的复杂性:C2~C4,C10,C11.建模是软件设计的一项重要活动,它决定了子系统的结构、接口、

对象等.例如,C2 就与子系统结构的决策相关.设计决策是软件设计的基础,因此也存在许多相关研究,比如文献[66–69].尽管这些挑战背后的设计决策具有一定的特殊性,但一些观点仍然可以为实践者提供参考.比如在决策过程中,理性分析和主观经验都需要得到重视[27].具体而言,前者是根据预先定义的标准对备选决策进行评估和选择,比如应用 DDD 战术设计模式进行实体与值对象的划分时就需要理性分析;而后者则是根据经验形成令人满意的解决方案,比如在应用 DDD 战略设计模式来划分限界上下文时,无法完全脱离设计者的主观经验.因此,本文建议实践者在应对建模问题时既要进行理性决策,也要采纳一些合理的主观经验;

- 理解领域的困难:C1,C7,C9,C15.领域驱动设计要求在建模时与领域专家进行紧密配合[1],然而在具体实践中,想要让领域专家持续地参与建模过程并非易事,这需要软件开发组织为领域专家增加一笔额外的开销[4].此外,即使有领域专家的参与,开发团队仍需要面对与领域专家进行沟通(如 C7 和 C15)和领域认知(C9)方面所存在的挑战.本文建议实践者增加对于理解领域这一过程的投入,并应用一些创新方法,如尝试从多个角度(M12)来管理复杂性;
- 对于除领域外的其他方面关注过少:C5,C12,C13.这些挑战的重点在于,DDD 强调通过关注业务领域的建模来解决软件开发的复杂性问题,因此,DDDP 也主要涉及建模方面,而缺少对其他方面(比如实现)的明确指导.上述挑战可以通过补充工件或更加完善的规范来解决,比如领域层之外的工件(M6)和来自 MDD 的中间模型(M14)等;
- DDD 理论的成熟度欠佳:C6,C8,C14,C16,C17.无论是通用语言还是战略和战术设计模式,都是 DDD 所提供的抽象指导,实践者仍然缺乏应用这些 DDDP 进行软件开发所需要的工具、方法以及过程管理等成熟能力.因此,在 DDD 领域未来的研究工作中,可以考虑通过提出创新性解决方案(如 M7)或者参考其他软件工程社区中已有方法(如 M17),为将 DDDP 规范化地应用于软件开发过程提供更完善的支持.

Table 10 Classification of the challenges of applying DDD patterns

表 10 应用 DDDP 的挑战分类

分类描述	DDD 挑战
建模本身的复杂性	C2~C4,C10,C11
理解领域的困难	C1,C7,C9,C15
对于除领域外的其他方面关注过少	C5,C12,C13
DDD 理论的成熟度欠佳	C6,C8,C14,C16,C17

为便于理解,本文将这 4 类挑战对应到领域驱动设计方法及其模式在软件开发中扮演的角色,如图 7 所示.

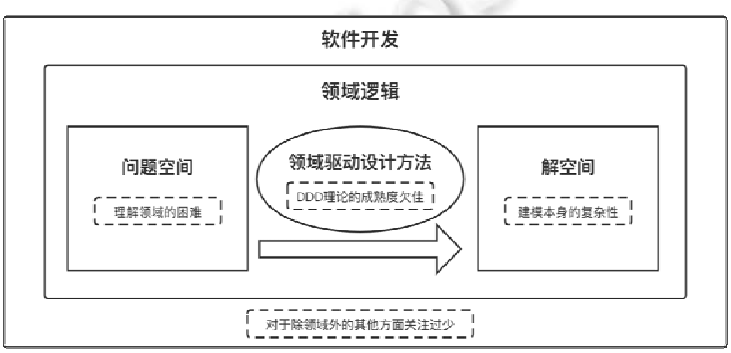


Fig.7 Relation ship between four categories of challenges

图 7 4 类挑战的关系

作为一种软件设计方法,领域驱动设计旨在加速具有复杂领域逻辑的软件项目开发[1].领域驱动设计利用一系列 DDDP 从领域逻辑中抽象领域模型的过程,从本质上看,可以理解为用户应用软件的领域(即问题空间)到可运行的软件(即解空间)之间的映射和转换.这个过程大体上可以分为两个阶段:对问题空间的理解和基于

对问题空间的理解设计解空间.相应地,应用 DDDP 面临着理解问题空间时“理解领域的困难”挑战以及基于对问题空间的理解设计解空间时“建模本身的复杂性”挑战.另外,这个过程还面临着“DDD 理论的成熟度欠佳”挑战.由于软件开发除领域逻辑外还需解决技术等其他难题,在实际软件项目开发中落地 DDDP 还面临着“对于除领域外的其他方面关注较少”挑战.

对于应对实践 DDDP 时所面临挑战的缓解方法而言,既有积极的一面,也有消极的一面.

- 在积极的一面,这些缓解方法代表了实践者为应对 DDDP 所带来的挑战而做的努力:首先,一些缓解方法代表了 DDD 实践者当前的最佳实践,比如利用事件风暴(event storming)来探索领域(M4)和建模规约(modeling conventions)(M13);其次,为了更好地应用 DDDP,领域驱动设计的实践者们付出大量努力,包括提出 UML 配置文件(M7)和元模型(M8)等解决方案;最后,一些缓解方法中展现了从其他方法论中借鉴来的智慧,如中间模型(M14)和领域视图(M12)等;
- 在消极的一面,DDDP 的应用中仍然存在很多问题:首先,尽管针对 C6 的缓解方法,也就是 UML 配置文件(M7)和元模型(M8)具有一定效果,但它们违背了 Evans 的初衷[30],具体而言,虽然 M7 和 M8 可以在一定程度上使领域模型标准化,并减少歧义,但由于它们增加了更多的规则约束,所以也降低建模过程的创造性;除此之外,一些缓解方法仅仅根据案例研究进行了评估,还缺少进一步的验证,比如 C8 的建模工具和软件开发过程概览(M17);最后,一些缓解方法本身并不容易实施,因为这些方法既没有给出系统性的指导,也没有给出具体实施的提示.

4.4 对研究者的启示

在目前关于 DDDP 的学术研究中,已发表研究文献的主要关注点是对 DDDP 详细内容的完善和在软件项目中应用 DDDP 两类,主要贡献也集中于提出新的方法或者总结应用 DDDP 的实践经验.然而,在软件工程中是“没有银弹”的,任何理论方法都会存在一定的局限性和缺点,DDDP 也是如此.相比于已经广泛应用 DDD 理论的软件工业界而言,对于 DDDP 应用的反思,在学术研究文献中还比较少见.

此外,DDD 应用实践时除了需要掌握以领域为中心的思考方式之外,还应该能够熟练地掌握正确应用包括战略和战术设计方面在内的各种 DDDP.然而,相关研究文献以及著作中所论述的 DDDP 应用策略都比较抽象,这使得在实际情况下应用 DDDP 还缺少系统性的指导方法和良好实践,DDDP 的应用在很大程度上仍依赖于主观经验.因此,未来研究者可以考虑细化应用 DDDP 的指导方法,并提出或总结更多可供参考的良好实践.

5 效度威胁讨论

本系统文献综述的工作中存在的效度威胁来源于文献筛选、数据抽取及数据合成过程中可能存在的偏见.

为了消除检索字符串及数据库选择中存在的偏见,我们在自动检索之前进行了手动检索并确保:① 手动检索中出现的每个 DDD 同义词都已被合并到检索字符串中;② 手动检索获得的每篇文献都包含在自动检索结果集合中.之后,我们通过滚雪球过程对于每个 DDDP 所执行的单独自动检索结果进行补充.此外,本研究还预定义了一个已知的论文集合,以检查检索过程的全面性和准确性.为了避免文献筛选出现错误,每篇文献均由至少两名作者独立筛选与评估,然后通过二次审核过程进行合并.在这个过程中,我们使用了 Kappa 分析确保足够的可靠性[18].

在数据抽取与合成部分,效度威胁主要源于抽取过程中可能遗漏了有价值的文本数据,从而导致合成过程得出不准确的结论.为了避免遗漏有价值的文本数据,本研究在进行数据抽取过程之前先进行了小规模实验,并按照作者的理解对数据抽取形式进行了调整.基于实验调整后的数据项形式,两名作者独立地对每篇研究文献进行数据抽取,并在最后进行合并.为了减少数据合成过程中得出的不准确的结论,作者们首先对 DDD 相关书籍[1,4,18,70−72]进行学习和探讨,从而熟悉 DDD 社区.之后,两名作者独立进行编码过程,并与第三名作者协商解决这一过程中存在的冲突或疑问.同时,在本研究中通过对于编码结果的两次复核,保证了对结果的改进.

6 总结与展望

本文面向领域驱动设计模式这一主题进行了系统文献综述,通过对 2003 年~2019 年 7 月的研究文献进行识别、检索、筛选和分析,最终综合 26 篇高质量基础研究来回答所本文所提出的研究问题.本文提供了学术界对于 DDDP 应用的研究现状概览,同时比较全面地综述了基础研究中应用 DDDP 带来的收益、挑战以及应对挑战的缓解方法.

一方面,本文的研究表明:由于重视领域知识,实践者往往能够借助 DDDP 设计出良好的软件,并享受到其所带来的收益,如架构质量改善和代码意图清晰.另一方面,本文的数据综合结果表明:46.4%的基础研究认为在 DDDP 的实践中存在很多困难,并将其总结为 17 个挑战.本研究中还发现了 17 种缓解方法,用于应对上述提到的部分挑战.虽然这些缓解方法还不足以完美地支撑 DDDP 的应用实践,但是它们代表了实践者所付出的努力.正是这些该领域的理论和研究的不完善之处,为研究者和实践者们提供了更加广阔的探索空间.

目前,应用领域驱动设计模式在实践中的价值得到了比较广泛的认可,但对于如何更好地享受其所带来的收益,以及更加合理地应对其带来的挑战,则需要研究者进一步探索.与此同时,在软件工程中,没有任何一种技术能够称为“银弹”,因此,应用各种 DDDP 所带来的局限或者挑战,仍需要未来进一步探索和反思.

References:

[1] Evans E, Wrote; Zhao L, Sheng HY, Liu X *et al*., Trans. Domain-driven Design: Tackling Complexity in the Heart of Software. Beijing: Posts & Telecom Press, 2003 (in Chinese).

[2] Newman S, Wrote; Cui LQ, Zhang J, Trans. Domain-driven Design Quickly. Beijing: Posts & Telecom Press, 2016. (in Chinese).

[3] Nilsson J. Applying Domain-driven Design and Patterns: With Examples in C# and. NET. Pearson Education, 2006. 28−44.

[4] Vernon V. Implementing Domain-driven Design.1st ed. Addison-Wesley, 2013. 28−82.

[5] Millett S, Tune N. Patterns, Principles, and Practices of Domain-driven Design. John Wiley & Sons, 2015. 17−22.

[6] Bruhiere X. Use domain-driven design to architect your cloud apps. IBM, 2018. https://developer.ibm.com/tutorials/cl-domain-driven-design-event-sourcing

[7] Alibaba T. Creating coding excellence with domain-driven design. Medium, 2018. https://medium.com/swlh/creating-coding-excellence-with-domain-driven-design-88f73d2232c3

[8] Newman S. Building Microservices: Designing Fine-grained Systems. O’Reilly Media, Inc., 2015. 1−8.

[9] Rademacher F, Sorgalla J, Sachweh S. Challenges of domain-driven microservice design: A model-driven perspective. IEEE Software, 2018,35(3):36−43.

[10] Zhang H, Li S, Jia Z, Zhong C, Zhang C. Microservice architecture in reality: An industrial inquiry. In: Proc. of the 2019 IEEE Int’l Conf. on Software Architecture (ICSA). IEEE, 2019. 51−60.

[11] Singh M, Sharma A, Saxena R. Formal transformation of uml diagram: Use case, class, sequence diagram with z notation for representing the static and dynamic perspectives of system. In: Proc. of the Int’l Conf. on ICT for Sustainable Development. Singapore: Springer-Verlag, 2016. 25−38.

[12] Bogner J, Fritzsch J, Wagner S, Zimmermann A. Microservices in industry: Insights into technologies, characteristics, and software quality. In: Proc. of the 2019 IEEE Int’l Conf. on Software Architecture Companion (ICSA-C). Hamburg: IEEE, 2019. 187−195.

[13] Jorgensen M, Shepperd M. Asystematic review of software development cost estimation studies. IEEE Trans. on Software Engineering, 2007,33(1):33−53.

[14] Budgen D, Burn AJ, Brereton OP, Kitchenham BA, Pretorius R. Empirical evidence about the UML: A systematic literature review. Software: Practice and Experience, 2011,41(4):363−392.

[15] Shahin M, Babar MA, Zhu L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 2017,5:3909−3943.

[16] Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley Longman Publishing Co., Inc., 2002. 22−31.

[17] Larman C. Agile and Iterative Development: A Manager’s Guide. Addison-Wesley Professional, 2004. 128−140.

[18] Kitchenham BA, Budgen D, Brereton P. Evidence-Based Software Engineering and Systematic Reviews. CRC Press, 2015. 3−9.

[19] Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of the 18th Int'l Conf. on Evaluation and Assessment in Software Engineering. New York: ACM, 2014. 1–10.

[20] Evans E. Domain-driven Design Reference: Definitions and Pattern Summaries. Dog Ear Publishing, 2014. 15–30.

[21] Cohen J. Weightedkappa: Nominalscaleagreementprovision for scaled disagreement or partial credit. Psychological Bulletin, 1968, 70(4):213.

[22] Dybå T, Dingsøyr T. Empirical studies of agile software development: A systematic review. Information and Software Technology, 2008,50(9-10):833–859.

[23] Zhang C, Budgen D. What do we know about the effectiveness of software design patterns? IEEE Trans. on Software Engineering, 2011,38(5):1213–1231.

[24] Cruzes DS, Dyba T. Recommended steps for the matic synthesis in software engineering. In: Proc. of the 2011 Int'l Symp. on Empirical Software Engineering and Measurement. Banff: IEEE, 2011. 275–284.

[25] Roman D, Keller U, Lausen H, DeBruijn J, Lara R, Stollberg M, Polleres A, Feier C, Bussler C, Fensel D. Web service modeling ontology. Applied Ontology, 2005,1:77–106.

[26] Charmaz K, Belgrave LL. Grounded Theory. The Blackwell Encyclopedia of Sociology, 2007.

[27] Osborne JW. Best Practices in Quantitative Methods. Thousand Oaks: SAGE Publications, Inc., 2008. 1–9.

[28] Braun V, Clarke V. Using thematic analysis in psychology. Qualitative Research in Psychology, 2006,3(2):77–101.

[29] Haywood D. Domain-driven Design using Naked Objects. Pragmatic Bookshelf, 2009.

[30] Hippchen B, Giessler P, Steinegger R, Schneider M, Abeck S. Designing microservice-based applications by using a domain-driven design approach. Int'l Journal on Advances in Software, 2017,10(3&4):432–445.

[31] Bruegge B, Dutoit AH. Object-oriented Software Engineering using UML, patterns, and Java. Learning, 2009,5(6):7.

[32] Zandin M. Domain-driven design and microservices. InfoQ, 2016. https://www.infoq.com/news/2016/04/ddd-microservices-evans.

[33] Easterbrook S, Singer J, Storey MA, Damian D. Selecting empirical methods for software engineering research. In: Proc. of the Guide to Advanced Empirical Software Engineering. London: Springer-Verlag, 2008. 285–311.

[34] Vural H, Koyuncu M, Misra S. A case study on measuring the size of microservices. In: Proc. of the Int'l Conf. on Computational Science and Its Applications. Cham: Springer-Verlag, 2018. 454–463.

[35] Lotz G, Peters T, Zrenner E, Wilke R. A domain model of a clinical reading center-design and implementation. In: Proc. of the 2010 Annual Int'l Conf. of the IEEE Engineering in Medicine and Biology. Buenos Aires: IEEE, 2010. 4530–4533.

[36] Santos ECS, Beder DM, Penteado RAD. A Study of test techniques for integration with domain driven design. In: Proc. of the 2015 12th Int'l Conf. on Information Technology—New Generations. Las Vegas: IEEE, 2015. 373–378.

[37] Landre E, Wesenberg H, Olmheim J. Agile enterprise software development using domain-driven design and test first. In: Proc. of the Companion to the 22nd ACM SIGPLAN Conf. on Object-oriented Programming Systems and Applications Companion. New York: ACM, 2007. 983–993.

[38] Salahat M, Wade S. Pedagogical evaluation of a systemic soft domain-driven design framework. In: Proc. of the UKAIS Int'l Conf. on Information Systems. 2012. 48.

[39] Danenas P, Garsva G. Domain driven development and feature driven development for development of decision support systems. In: Proc. of the Int'l Conf. on Information and Software Technologies. Berlin: Springer-Verlag, 2012. 187–198.

[40] Schneider M, Hippchen B, Abeck S, Jacoby M, Herzog R. Enabling IoT platform interoperability using a systematic development approach by example. In: Proc. of the 2018 Global Internet of Things Summit. Bilbao: IEEE, 2018. 1–6.

[41] Munezero IJ, Mukasa DT, Kanagwa B, Kanagwa, Balikuddembe J. Partitioning microservices: A domain engineering approach. In: Proc. of the 2018 IEEE/ACM Symp. on Software Engineering in Africa. Gothenburg: IEEE, 2018. 43–49.

[42] Uludağ Ö, Hauder M, Kleehaus M, Schimpfle C, Matthes F. Supporting large-scale agile development with domain-driven design. In: Proc. of the Int'l Conf. on Agile Software Development. Cham: Springer-Verlag, 2018. 232–247.

[43] Bruyn P D , Huysmans P , Mannaert H . Tailoring an Analysis Approach for Developing Evolvable Software Systems: Experiences from Three Case Studies. In: Proc. Of the 18th IEEE Conference on Business Informatics. Paris: IEEE, 2016. 208–217.

[44] Rademacher F, Sachweh S, Zündorf A. Towards a UML profile for domain-driven design of microservice architectures. In: Proc. of the Int'l Conf. on Software Engineering and Formal Methods. Cham: Springer-Verlag, 2017. 230–245.

[45] Razavi R. Web pontoon: A method for reflective web applications. In: Proc. of the Int'l Workshop on Smalltalk Technologies. New York: ACM, 2010. 1–10.

[46] Salahat M, Wade S. Domain driven design vs soft domain driven design frameworks. Int'l Journal of Computer and Systems Engineering, 2016,10(7):1364–1370.

[47] Diepenbrock A, Rademacher F, Sachweh S. An ontology-based approach for domain-driven design of microservice architectures. In: Proc. of the INFORMATIK 2017. Bonn: Gesellschaft für Informatik, 2017. 1777–1791.

[48] Rademacher F, Sorgalla J, Sachweh S. Challenges of domain-driven microservice design: A model-driven perspective. IEEE Software, 2018,35(3):36–43.

[49] Sousa NT, Hasselbring W, Weber T, Kranzlmu□ D. Designing a generic research data infrastructure architecture with continuous software engineering. In: Combined Proc. of the Workshops of the German Software Engineering Conf. 2018. 85–88.

[50] Pham M, Hoang DB. SDN applications—The Intent-based northbound interface realisation for extended applications. In: Proc. of the 2016 IEEE NetSoft Conf. and Workshops. Seoul: IEEE, 2016. 372–377.

[51] Shadija D, Rezai M, Hill R. Towards an understanding of microservices. In: Proc. of the 2017 23rd Int'l Conf. on Automation and Computing. Huddersfield: IEEE, 2017. 1–6.

[52] Landre E, Wesenberg H, Rønneberg H. Architectural improvement by use of strategic level domain-driven design. In: Proc. of the Companion to the 21st ACM SIGPLAN Symp. on Object-oriented Programming Systems, Languages, and Applications. New York: ACM, 2006. 809–814.

[53] Wesenberg H, Landre E, Rønneberg H. Using domain-driven design to evaluate commercial off-the-shelf software. In: Proc. of the Companion to the 21st ACM SIGPLAN Symp. on Object-oriented Programming Systems, Languages, and Applications. New York: ACM, 2006. 824–829.

[54] Zhang J, Chen Y, Qin S. The application of domain-driven design in NMS. In: Proc. of the 4th Int'l Conf. on Machine Vision. Singapore, 2012.

[55] Peng S, Hu Y. IAnticorruption: A domain-driven design approach to more robust integration. In: Proc. of the Companion to the 22nd Annual. New York: ACM, 2007. 976–982.

[56] Soares SA, Cortés MI. Elihu: A project to model-driven development with naked objects and domain-driven design. In: Proc. of the 20th Int'l Conf. on Enterprise Information Systems. 2018. 272–279.

[57] Koryl M. Active resources concept of computation for enterprise software. Archives of Control Sciences, 2017,27(2): 279–291.

[58] Debski A, Szczepanik B, Malawski M, Spahr S, Muthig D. A scalable, reactive architecture for cloud applications. IEEE Software, 2017,35(2):62–71.

[59] Soldani J, Tamburri DA, Van Den Heuvel WJ. The pains and gains of microservices: A systematic grey literature review. Journal of Systems and Software, 2018,146:215–232.

[60] Brandolini A. Introducing Event Storming. Lean Publishing, 2013. 21–24.

[61] Fairbanks G. Just Enough Software Architecture: A Risk-driven Approach. 3rd ed., Marshall & Brainerd, 2010. 15–32.

[62] Staab S, Studer R. Handbook on Ontologies. 2nd ed., Springer-Verlag Science & Business Media, 2010. 1–9.

[63] Selic B. The pragmatics of model-driven development. IEEE Software, 2003,20(5):19–25.

[64] Rosenfeld L, Morville P, Nielsen J. Information Architecture for the World Wide Web. 2nd ed., O'Reilly Media, Inc., 2002. 3–5.

[65] Greg Y. CQRS and event sourcing. CodeBetter, 2010. http://codebetter.com/gregyoung/2010/02/13/cqrs-and-event-sourcing.

[66] Conklin J, Begeman ML. gIBIS: A hypertext tool for exploratory policy discussion. ACM Trans. on Information Systems (TOIS), 1988,6(4):303–331.

[67] Burge JE. Software engineering using design rationale [Ph.D. Thesis]. Massachusetts: Worcester Polytechnic Institute, 2005.

[68] Van VH, Tang A. Decision making in software architecture. Journal of Systems and Software, 2016,117:638–644.

[69] Lemma R, Lanza M, Mocci A. Cel: Touching software modeling in essence. In: Proc. of the 2015 IEEE 22nd Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Montreal: IEEE, 2015. 439–448.

[70] Kitchenham B, Brereton P. A systematic review of systematic review process research in software engineering. Information and Software Technology, 2013,55(12):2049–2075.

[71] Kitchenham B, Dyba T, Jorgensen M. Evidence-based software engineering. In: Proc. of the 26th Int'l Conf. on Software Engineering. Edinburgh: IEEE Computer Society, 2004. 273–281.

[72] Faraj S, Sproull L. Coordinating expertise in software development teams. Management Science, 2000,46(12):1554–1568.

附中文参考文献:

[1] Evans E,著;赵俐,盛海艳,刘霞,等译.领域驱动设计——软件核心复杂性应对之道.北京:人民邮电出版社,2003.

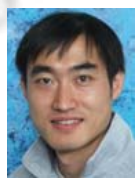
[2] Newman S,著;崔力强,张骏,译.微服务设计.北京:人民邮电出版社,2016.

贾子甲(1994－),男,硕士,CCF 学生会员,主要研究领域为软件架构,微服务,领域驱动设计.

钟陈星(1995－),女,学士,CCF 学生会员,主要研究领域为微服务,领域驱动设计.

周世旗(1998－),女,硕士,主要研究领域为软件架构.

荣国平(1977－),男,博士,副研究员,CCF 专业会员,主要研究领域为软件过程实证软件工程.



章程(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为微服务,软件体系结构,经验软件工程.