软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(10):3127−3147 [doi: 10.13328/j.cnki.jos.005792]

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

面向完整价值交付的文档 DevOps 应用研究*

金泽锋[1], 张佑文[1], 叶文华[1], 张 贺[2], 邵 栋[2]

[1](中兴通讯,江苏 南京 210012)

[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)

通讯作者: 金泽锋, E-mail: jin.zefeng@zte.com.cn

摘 要: 随着 DevOps 在各大软件企业中的广泛实施,加速了系统软件类产品的版本交付和部署.中兴通讯在实施过程中,发现产品重要的组成部分——产品文档,还采用传统研发方式,缺乏配套流程和工具,导致产品文档交付的节奏无法与软件版本匹配,严重影响了产品完整交付的及时性.文档 DevOps 是一种针对产品文档持续交付的模式.通过分析开源和 DITA 的文档交付解决方案,同时结合系统软件的研发特点,总结出一套适合系统软件的文档交付综合解决方案,它借助 DevOps 理念,基于业界的 DevOps 工具链,构建了“文档 DevOps 平台”(以下简称 iDoc 平台),实现面向用户各类文档交付的解决方案,其典型特征有:与软件迭代流程融合,信息单元同源引用,多格式内容源的管理,持续集成的质量守护.在实际企业中通过文档 DevOps 实现了产品文档与软件版本的同步交付,极大地促进了文档的准确性、完整性和交付效率的提升.iDoc 平台已在 50 多个软件产品中得到成功应用.文档 DevOps 解决的问题具有普遍性,有助于在更大范围内帮助其他系统软件实现敏捷价值交付,并且,文档 DevOps 还是对 DevOps 的一个补充,扩展了业界的 DevOps 的适用范围:交付对象涵盖了产品文档,流程延伸到市场规划,协同人员覆盖面更广.

关键词: 敏捷;DevOps;持续集成;持续交付;文档;文档 DevOps;价值交付

中图法分类号: TP311

中文引用格式: 金泽锋,张佑文,叶文华,张贺,邵栋.面向完整价值交付的文档 DevOps 应用研究.软件学报,2019,30(10):3127−3147. http://www.jos.org.cn/1000-9825/5792.htm

英文引用格式: Jin ZF, Zhang YW, YE WH, Zhang H, Shao D. Research on application of DevOps in documentation towards full value delivery. Ruan Jian Xue Bao/Journal of Software, 2019,30(10):3127−3147 (in Chinese). http://www.jos.org.cn/1000-9825/5792.htm

Research on Application of DevOps in Documentation towards Full Value Delivery

JIN Ze-Feng[1], ZHANG You-Wen[1], YE Wen-Hua[1], ZHANG He[2], SHAO Dong[2]

[1](Zhongxing Telecommunication Equipment Corporation, Nanjing 210012, China)

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

Abstract: With the extensive implementation of DevOps in major software enterprises, the version delivery and deployment of system software class products has been accelerated. In the process of implementation, it is found that the important part of the product, the product document, is still developed by traditional R&D method and lack of supporting processes and tools, which leads to the delay in the product document delivery comparing with the delivery of the software version, and which seriously affected the complete and just-in-time delivery of the product. The “Document DevOps” introduced in this paper is a continuous product document delivery mode. By analyzing the document delivery solutions of open source and DITA, and combining with the R&D characteristics of system software, a set of comprehensive document delivery solutions that is fit for system software has been summarized. With the concept of DevOps, and

* 基金项目: 国家自然科学基金(61572251)
Foundation item: National Natural Science Foundation of China (61572251)
本文由“面向 DevOps 的软件工程新技术”专题特约编辑荣国平、白晓颖、岳涛推荐.
收稿时间: 2018-08-31; 修改时间: 2018-10-31; 采用时间: 2018-12-14; jos 在线出版时间: 2019-08-09
CNKI 网络优先出版: 2019-08-12 12:08:09, http://kns.cnki.net/kcms/detail/11.2560.TP.20190812.1207.005.html

based on the DevOps tool chain of the industry, it has contracted a “Document DevOps Platform” (hereinafter referred to as the platform), to achieve a user-oriented document delivery solution. Its typical characteristics are: integration with software iteration process, information unit homologous citation, management of multi-format content sources, and protection of continuous integration quality. Zhongxing has achieved the simultaneous delivery of product documents and software versions by the “Document DevOps”, which greatly improves the accuracy, integrity, and delivery efficiency of documents. The iDoc platform has been successfully applied in more than 50 software products. The problem solved by the “Document DevOps” is universal, helping the other system software to achieve agile value delivery in a wider range. Moreover, the “Document DevOps” is a supplement to DevOps, for it extends the application scope of DevOps: the delivery object covers product documentation, and the process extends to market planning, covering a broader range of collaborative personnel.

Key words: agile; DevOps; continuous integration; continuous delivery; document; document DevOps; value delivery

IT 调研与咨询服务公司 Gartner 2015 的调查报告显示,全球前 2 000 家大型软件公司中 25%使用 DevOps 方式作为公司主流方式交付软件的模式[1].全球云管理平台领导者 RightScale 公司调查报告也显示,DevOps 已成为研发基于云的应用程序的默认方法了,其接受度从 2015 年的 66%增至 2017 年的 78%,而其中,大型企业的采用率更高达 84%,且以其作为主流软件交付模式的占到 30%[2,3].

软件产品不仅仅包含可执行程序和支撑数据(本文统称为软件版本),还含有配套的文档集合(以下简称产品文档).特别是在大规模系统软件产品(以下简称“系统软件”)研发和交付领域,产品文档还占有非常重要的地位,探索相关问题的解决方案的需求也更为迫切[4].本文的文档 DevOps 即是在业界这个方向上的实践与总结.

系统软件是一类为解决复杂问题或为服务庞大客户群所开发的软件,往往由大型企业、组织和政府所最终拥有,对软件产品的要求高.同时,大规模系统软件的研发和最后的运营维护往往由不同团队完成,例如电信产品由少数电信设备商研发交付,却由运营商或第三方来承担开通、运营、维护和优化.这个过程中,运维人员的产品知识一般来源于设备商提供的产品相关文档,所以,在系统软件交付的同时,产品文档也作为重要交付物需要交付给客户.因此,在产品文档开发方面会有如下几个重要挑战.

(1) 文档内容覆盖要全面且系统详尽:交付运营的特性需要进行大范围的产品知识转移,转移过程无法长期依靠研发专家的人工支持,而且高端客户往往都是学习型组织,对大规模软件系统的了解非常深入,需要大量对细节的描述.这些必须依赖于系统、详尽的产品文档,便于规模化知识的转移.

(2) 软件和文档的高质量要求:通常,大规模软件系统的最终用户(例如 ISP 的客户)的数量非常庞大,倘若出现故障,一般影响面都会很大,这就要求软件系统本身应该具备高可靠性,同时运维人员的操作需严格遵循操作说明执行,以便提供稳定的运维服务.这意味着不仅交付的软件系统应该包含尽可能少的缺陷,而且相应的产品文档中的内容应该完全和产品保持一样的高质量要求,同时必须和软件产品保持完全一致.例如,文档内容中通常未明确说明其描述的软件版本,读者不清楚所提供的信息是否是最新的,而文档描述的内容在不同版本之间往往是有明显差异的[5].

(3) 要完整表述高复杂的系统,文档架构也必然复杂:通常交付给客户的产品错综复杂,而且使用人员群体庞大、个体差异明显,这两个方面最终导致产品文档内容通常非常庞大而且复杂.不同产品间的文档还需要关注相互的引用要准确,表述要规范、清晰.

综上所述,产品配套文档应具有以下“四性”.

(1) 准确性:产品文档描述的内容,与实际设备的内外部特征和使用方法保持一致,并反映产品的最新更新.

(2) 完整性:产品文档中能够系统、详尽地描述产品所提供的全部功能,并包含产品开通、运营、维护和优化等各类活动涉及到的产品相关知识.

(3) 及时性:产品文档需要在使用时间前完全准备好,用户可以在需要的时候快速获取到产品的配套文档.

(4) 可用性:文档结构清晰,关键内容突出,步骤详细,可以指导目标用户高效地完成工作.同时,在各部分文件间的内容描述是规范的,用户体验是一致的.

实际系统软件交付过程中,产品文档大量存在着内容不准确、不一致[6],内容缺失[7,8],交付严重滞后[9,10]的现象,与用户对文档的要求存在较大的差距.通过产业界和学术界的合作,参考 DevOps 的理念、实践和工具,我

们设计了一种新的文档开发模式“文档 DevOps”和配套的文档 DevOps 平台“iDOC”来解决以上问题.

文档 DevOps 首先将文档代码化,将文档的开发过程内置到敏捷流程中,与需求开发交付过程匹配进度,并且利用文档模板解耦内容,结合同源工具生产后续面向各类用户的产品文档,以实现在全程消除内容的冗余.文档 DevOps 中还借鉴 CI(continuous integration,持续集成)技术,将面向文档的质量检查工具内置在日常的文档编写集成过程,做到快速反馈和质量内建.文档 DevOps 也和软件版本的 DevOps 一样,打通了开发阶段后到发布阶段的反馈环,并能够做到基于版本管理的自动化发布.

本文文档 DevOps 模式中提供了的相关的流程与实现架构,能够让软件版本和产品文档同步发布,并且利用 DevOps 的优势提升了文档开发协作效率和产品文档的质量.

本文工作的意义在于完善了软件工程中关于 DevOps 的完整性描述,涵盖了版本和文档两类主要对象.同时,有力地支持了敏捷中价值交付的实现:用户的价值诉求不光是软件版本,还有产品文档,利用敏捷迭代对客户完整交付系统软件具有现实意义.

1 背景介绍

1.1 问题产生

ZTE 是全球顶级的电信企业,拥有超过 30 000 名研发人员,为互联网服务提供商和电信集团提供硬件设备和大规模软件系统.

如图 1 所示,ZTE 在 2013 年前使用传统流程(即瀑布模型)来开发和交付软件产品,大规模系统软件的版本交付周期平均约为 9~12 个月.

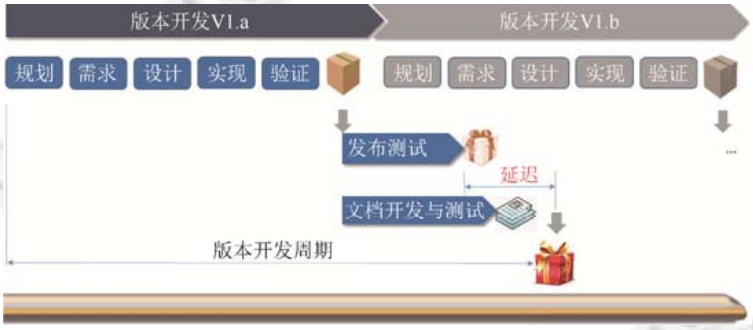

Fig.1 The traditional way to deliver a version

图 1 传统产品交付模型

在传统交付流程中,产品文档开发往往从版本的发布测试阶段开始.技术写作者不直接参与产品研发过程,而仅是后期从事产品文档的开发.他们所需要的产品知识是依赖于研发人员输出的产品文档素材或者和研发人员沟通确认来获取必要的信息.这个串行开发过程有如下几个缺点.

(1) 导致文档交付滞后于软件版本,影响到产品文档交付的及时性.系统软件的文档交付周期经常延迟 2 个月以上.

(2) 技术写作者与研发人员通过文档素材抛接传递产品知识,会严重影响文档的准确性和完整性.其原因是面对研发人员提供的素材,技术写作者在双方知识背景偏差较大时容易产生理解偏差.时间点上,素材的沟通确认与产品功能的研发之间往往隔了几个月,会加剧理解和沟通的难度.

(3) 质量反馈周期过长,文档开发改进慢.批量交付模式导致问题的反馈周期长,又集中在产品交付后期,交付压力下文档测试容易被压缩,测试不充分,文档准确性、完整性、可用性等方面故障泄露到用户.当前发现的文档问题,相应的研发人员正在忙其他需求,参与解决问题的成本高.大量文档问题被积压后,常依赖运动式质量提升活动解决,针对文档开发过程的改进频次低且沉淀不足.

2013 年在该公司内部为应对外部市场变化,针对软件研发发起了一场敏捷变革,敏捷的软件版本交付模式如图 2 所示.

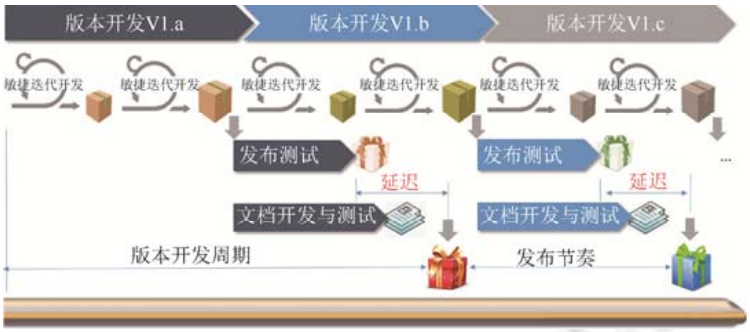

Fig.2 The agile way to deliver a version

图 2 敏捷软件版本交付模型

研发团队采用敏捷的迭代流程完成高频、小批量的需求交付,并采用 DevOps 进一步加快功能开发、集成和软件版本交付的速度.与传统产品交付模型相比,发布节奏平均缩短到 1~6 个月.

但在敏捷软件交付模型中忽略了产品文档开发的配套改进,敏捷优秀实践的引入让如下问题变得更加突出.

(1) 版本频繁发布带来配套文档延迟的频度变得更多了,文档同步交付的及时性需求更加迫切.

(2) 敏捷鼓励面对面沟通,轻量级文档导致技术写作者的工作素材来源不足[11,12].敏捷宣言中“可以工作的软件胜过面面俱到的文档”[13]往往被误解[14],软件开发过程的内部文档甚至被一些敏捷专家视为浪费[15],这种误导也进一步弱化了文档素材的及时输出.在很多敏捷项目中,文档变成了烫手山芋,无人愿意接手[16].

(3) 敏捷鼓励全栈工程师容易导致更为频繁的责任变更,技术写作者经常会遇到找不到合适的研发人员来获取文档相关信息的情况.

敏捷变革使得版本交付大幅提效,但产品文档交付的及时性和效率却遇到更大挑战,无法同步提供产品文档,产品文档的开发已成为整个组织交付改进的瓶颈.

1.2 行业方案分析

在大型软件系统的文档开发领域,业界有两类较为成熟的解决方案:以 DITA 为代表的专业文档编写和发布方案和以 Openstack 社区为代表的开源文档代码化的解决方案.这两个方案各有适用场景和特点,分析如下.

1.2.1 大型产品的专业文档编写和发布解决方案

DITA(darwin information typing architecture)是 OASIS 技术标准,它利用 XML 技术框架,构建了一套完整的用于文档的方案.DITA 的解决方案支持模块化写作,定义了标准的信息模块,如基本 Topic、Concept、Task、Reference 这 4 种 Topic,同时,DITA 也支持定义新的信息模块(topic)[17,18].标准化的模块有助于分工协作写作,也便于信息的复用,相同信息只需写作 1 次,便可应用在各类文档中.例如一个 Topic 的信息,可以同时用于“特性指导书”“用户手册”“联机帮助”等不同类型文档中.避免信息冗余,提高了信息开发效率,同时也降低了开发成本.DITA 支持内容与格式分离,基于 DITA 的写作可以实现内容与格式的分离,DITA 信息可以方便转换为各种格式.

对于系统软件产品团队,一般是采用定制的 DITA 商业工具,DITA 方案有两个比较显著的特征.

(1) 面向的是专业的技术写作者

DITA 相对比较复杂,在最新的 V1.3 全集中,Topic 类型有 26 种,编写元素高达 621 个,即便是在基本集中,Topic 类型有 4 种 Topic 类型,189 个编写元素[19].作为一种神秘、过于复杂的语言,学习曲线比较陡峭,对应的特殊编辑工具也比较专业,主要用于供技术写作者使用[20].由于这个原因,大量的 Topic 的内容需要由规划、研发等人员传递素材给技术写作者,技术写作者再进行加工和编写 Topic,不利于提升文档的准确性、及时性.

OASIS 也意识到 DITA 的复杂性,已经开始研究轻量级 DITA(lightweight DITA,简称 LwDITA),引入了 HTML5 格式(HDITA)和 MarkDown 格式(MDITA)[21].在目前开发尚未发布的 DITA2.0 版本中,重点也将减少 DITA 中大量未使用的特性.

(2) 重在解决产品文档间的冗余问题

DITA 方案更多地是考虑产品文档之间的信息同源与重用,文档的信息单元输出与软件系统的流程关联性很弱,导致产品文档开发和软件交付中的信息单元相互缺乏支撑,而且这种弱关联性也是文档与软件的不一致问题的重要来源[22].

总的来看,DITA 是比较专业和成熟的一套解决方案,在文档与软件的同步发布的挑战下,该方案的弱项在于没有考虑对敏捷模式的文档开发支撑,即如何将文档开发与软件开发的人员、信息和流程的协同整合,解决它们之间的内容一致性,发布同步性等问题.

1.2.2 开源社区的文档代码化解决方案

以 Openstack 为代表的开源社区,在开源软件 Openstack 的文档上探索了一条大型软件系统的文档代码化的解决方案[23].具有如下特征.

(1) 文档作者是开源社区的开发人员

开源社区中的软件团队往往缺少专业技术写作者,是由软件研发人员来编写文档内容.这种人员安排让信息的准确性有了基础,但要求开源文档的编写方式保持与软件的开发方式相一致,我们通常称这种方式为文档代码化.

开源社区文档的读者主要是研发人员,他们对产品文档的准确性更加看重.文档完整和准确与否,是开源软件被选用的决定性因素,所以在开源社区的文档解决方案中,比较关注产品文档与代码中信息的同源关系.

但开源文档的代码化解决方案中不涉及市场规划、工程服务、测试、文档活动中的人员,也不太涉及除代码以外的同源关系管理.

(2) 文档和代码流程统一管理

开源软件的变更活动和发布都比较频繁,对文档的频繁更新快速发布、文档与软件的一致性提出了更高的要求.

开源文档开发与软件开发基本一致:(a) 目前开源社区中,大量团队使用与代码编写方式非常类似的标记语言编写产品文档,如 Markdown、reStructedText 等.(b) 过程中拥有类似软件中的编码、检查、构建、发布等环节,并且基本上与软件节奏一致.(c) 引入了软件的持续集成、持续发布等实践和自动化工具.(d) 协作上,文档和代码都采用统一的存储方式,例如使用 Github 让代码和文档在同一个 repo 下,开发人员和文档工程师均可访问,版本控制采用与代码相同的 git 或 svn 这类工具进行版本管理.流程、技术工具和内容管理上的一致,较好地保证了软件与文档内容的一致性和发布的及时性[24].

开源解决方案的问题在于,开源组织比较松散,难以在全局整体信息内容上进行同源与重用的设计规划,不同人员的文档内容可能存在一定的重复或不一致.

1.2.3 小 结

综上对比分析可以看到,DITA 的解决方案难以确保敏捷交付模式下文档内容的“及时性”和“正确性”.开源解决方案没有考虑同源的设计,难以确保复杂系统文档的“准确性”和“完整性”.

在敏捷研发模式下,为了更好地解决大型软件系统的文档“四性”问题,还需要探索新的解决方案.

2 文档 DevOps 方案设计

2.1 本文解决方案

为了能够同步交付产品文档,可以将产品文档视为可持续交付的完整产品的有机部分、同步交付软件和文档.敏捷软件模式下的版本与文档交付模型如图 3 所示.

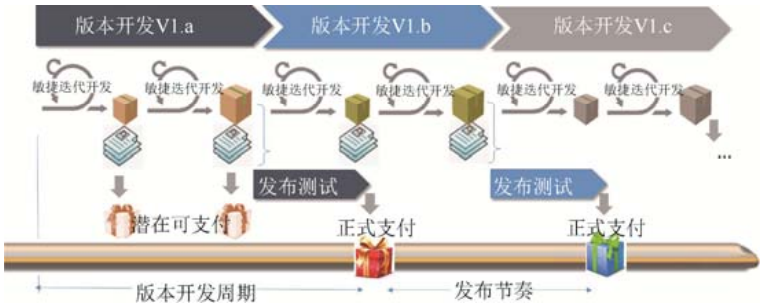

Fig.3 The agile way to deliver a product

图 3 敏捷产品交付模型

系统软件的研发是多个版本持续发布的过程.第 1 个迭代开发,到版本正式交付是版本的开发周期.敏捷软件开发中,每个迭代的交付中既包含了软件特性的潜在可交付,也包含了文档的潜在可交付,经过软件和文档的系统测试环节之后,就进入正式交付状态.基于该模型,文档的开发不再是一个单独的流程,而是与软件同步的迭代开发、持续交付的流程.

文档 DevOps 是基于 DevOps 理念的解决方案,通过文档开发的方法、流程和配套的支撑工具平台,实现大型软件系统各类文档的自动化构建与持续发布.在解决方案中,流动的基本对象是信息单元,它们按照组合形式和依赖关系的不同而有不同表现,共同构成文档 DevOps 的 3 层模型.

如图 4 所示,领域模型提供了文档 DevOps 的基础信息单元定义.通过文档架构的规划以及可以重用的同源信息单元的梳理,设计出各领域的不同类型的信息内容的模板,例如需求、方案、场景等,每个模型的模版中都包含大量的信息单元,如需求模型中包含“需求编号与标题”“背景”“预期收益”“验证准则”“相关波及”等.我们有一套名为“文档架构工作坊”的活动,用来完成文档架构规划和同源信息单元的梳理,其中通过规划、研发、工程等角色的协作分析,产生领域模型的定义与书写模板.研发过程中,各个角色再分别负责软件系统工作中与自己相关部分信息单元内容的输出,大量的信息单元可以有机地组合成为软件系统的文档.领域模型可以实现专业的人输出专业的信息内容,如高层设计 HLD(high level design)、特性描述 FD(feature description)、特性指导 FG(feature guide),保证了文档的准确性和完整性.而基于同源理念规划设计的信息单元,也给文档内容一致性提供了保障.

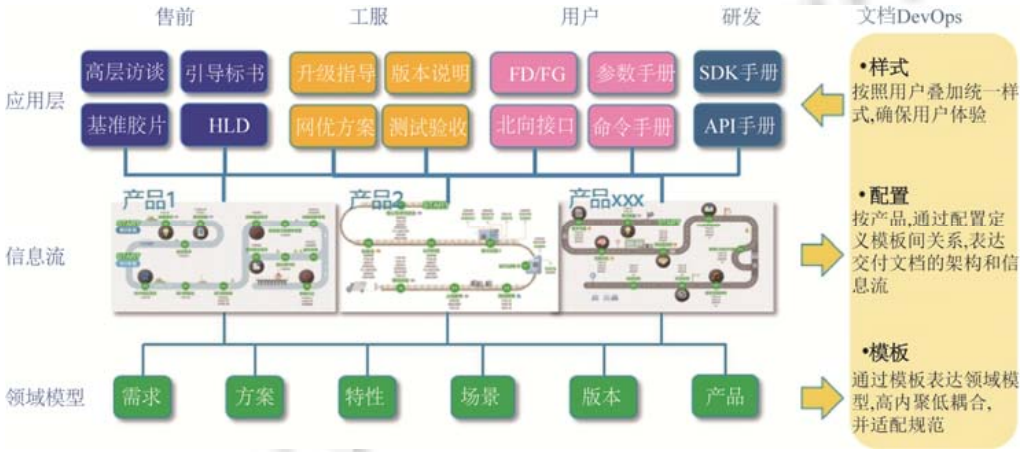

Fig.4 The model of document DevOps

图 4 文档 DevOps 模型

信息流定义了各类软件研发流程活动中需要输出的信息单元,并配置了这些信息单元如何组合成最终交付文档内容的映射关系,即信息单元的关联关系.具体而言,就是根据“文档架构工作坊”活动输出的软件系统文档架构,结合信息流中的信息单元及单元之间的引用、链接等关系,设计出文档的组合装配流水线(含应用在 DevOps 工具链的各类配置文件),就可以实现不同类型的文档的自动化构建、检查、组装、发布全过程.软件与文档融合的信息流的流程也保证了软件与文档的一致性,为文档发布的及时性提供了保障.

应用层根据文档面向的用户不同,可以提供不同的样式叠加,即在 DevOps 工具链编译完成内容准备后,再利用如 HTML 的 CSS 等技术,调整样式交付文档的表达格式.叠加样式前的内容,是由上一步的信息流输出的.整个样式叠加过程也是在 DevOps 工具链中完成,通过配置文件可以进行针对用户的需要进行内容的动态筛选和定制化输出.

根据以上分析,文档 DevOps 方案与业界常见的两种方案的对比见表 1.

Table 1 The comparison of solutions

表 1 解决方案比较

解决方案	侧重优化点	局限性
DITA 方案	分工协作 信息复用	面向专业技术写作者 与软件流程关联弱
开源代码化方案	轻量化标记语言编写与代码一致的实践和流程	代码化编写方式单一 缺少同源的设计规划
文档 DevOps 方案	同源规划与设计的信息单元 软件与文档流程融合 文档 DevOps 工具链 支持多格式的信息单元,以便每类信息生产者采用熟悉的方式输入	团队要熟悉和使用敏捷研发模式 已有 DevOps 实施的基础

其中,文档 DevOps 的收益来源于领域模型和信息流带来的同源效果,可同时为文档开发带来两种复用.

(1) 交付文档间的内容复用;

(2) 过程文档和交付文档的内容复用.

DITA 方案在第 1 点上是通过 Topic 在交付文件间复用;开源方案主要考虑第 2 点的优化,聚焦利用研发过程内容来输出最后的交付文档.本文的文档 DevOps 方案利用同源设计打通了交付文档间和过程文档间的内容引用关系,并将其自动化,同时获得了这两个方向上的收益.

我们在 60 多个电信软件项目中应用了文档 DevOps:软件版本和产品文档之间的滞后时间平均从 30 天左右缩短到不到 2 天,文档的一致性准确性问题也大幅下降,下降幅度达到 70%以上.

如图 5 所示,软件版本和产品文档发布节奏对齐后,文档 DevOps 扩展了传统 DevOps 的应用领域并支持连续的文档发布,实现了 DevOps 的全价值交付.

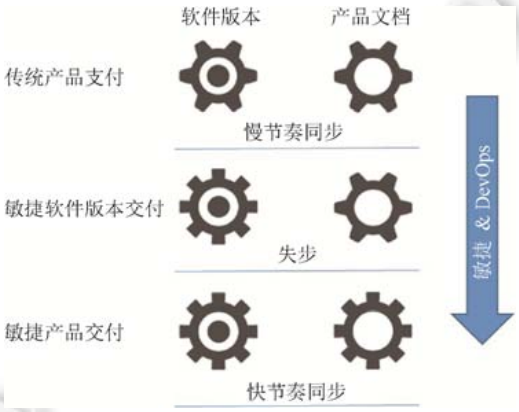

Fig.5 Different delivery paces of software and documents

图 5 软件版本和产品文档发布节奏对齐

2.2 业务流程

从文档交付业务流程来分析,文档 DevOps 有 4 个环节:生产、构建、发布和反馈,具体如图 6 所示.

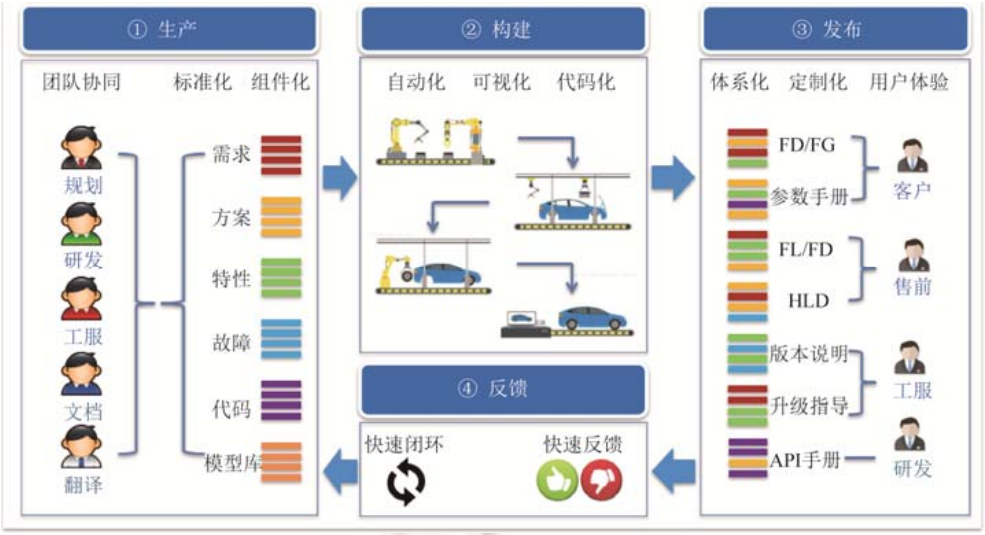

Fig.6 The process of document DevOps

图 6 文档 DevOps 的业务流程

2.2.1 生产环节

生产环节是文档的信息内容的产生过程,需要利用到分布式协同内容编写的工具,如 wiki 等,并在开发流程中内置好文档任务指派和自动化编译检查的功能.例如,在下发需求分析任务时,同步下发一个产品的特性指导手册的更新任务,将分析好的需求相关内容体现到产品特性指导手册中.生产环节有如下特性.

(1) 流程融合:生产活动贯穿于整个软件交付流程并与之融合,信息单元的输入、输出、承担的角色都依托于软件交付流程的输入输出及软件交付的角色,保证了文档与软件的节奏同步.

(2) 全员协同:软件系统的各个角色共同承担文档内容最终交付的职责,规划、研发、工服、技术写作者、翻译等输出的内容均要求达到可以对外交付的标准.信息单元由最熟悉内容的角色输出,准确性和专业性得到充分保障.例如,研发人员最熟悉他们开发的软件功能,直接由他们输出功能信息,可以有效避免知识转移过程中的失真.技术写作人员熟悉文档读者的阅读习惯和使用场景,可以站在用户角度测试最终文档,提升可读性、可用性和规范性.

(3) 组件化标准化:为了满足软件开发活动和外部文档的交付需要,同时最大化地降低冗余,需要对内容进行信息单元的可重用的组件化设计,采用模版进行标准化输出.与代码的模块化或函数的设计,以便代码可以更大范围地加以重用,信息单元的设计可以大大减少冗余的内容编写,降低文档编写的人力成本[25].信息单元既作为软件的输入,又是最终文档的构成部分,因而较好地实现了软件与文档的一致性,也满足了信息单元在多个文档的同源和一致性.

2.2.2 构建环节

构建环节是文档的信息内容的加工与文档合成.这个环节是当感知到内容编写工具中的信息单元内容变化并被提交保存后,根据存储在文档 DevOps 工具链中的文档架构配置文件(其中描述了信息单元和最终交付文档间的映射关系),触发自动流程,将以此信息单元为文档素材的相关交付文档重新进行组合和内容更新,并完成基本的文档质量检查.这个环节有如下特性.

(1) 代码化:基于代码的理念和实践,进行文档内容的信息单元的管理.如文档素材的信息单元用代码的管理工具 git 来管理;通过类似代码编译链接的工具,将不同格式和内容的信息单元加工、组合、打包为不同格式

的文档;通过类似静态代码检查的工具,实现文档内容的质量检查.代码化有助于把敏捷技术实践引入到文档开发中,支撑文档的敏捷开发,给文档与软件的同步发布提供了基础.

(2) 自动化:DevOps 倡导自动化一切,为高效交付、质量内建奠定基础.文档 DevOps 贯彻这一原则尽早、频繁、持续地自动化所有和文档开发相关的活动,包括采集、组合、打包等.

可视化:对文档的信息单元输出、自动化流程、质量、最终输出物的可视化,有助于整个团队了解文档的当前状态,充分暴露问题,促进成员团队的相互合作.

2.2.3 发布环节

发布环节是文档发布与获取的环节.这一环节需要用到在线的发布浏览系统,内部读者可以通过这个系统进行在线的评审与外发管理,外部读者可以通过系统查看和下载文档.此环节有如下特性.

(1) 体系化:各类文档以结构化方式来呈现和展示,用户可以根据文档体系快速浏览和查找到需要的内容.

(2) 定制化:文档传递的信息可以按照用户需要的格式和针对性内容来获取.大型系统软件的完整文档包体积达到 1GB 以上,不便于快速传递和精准查阅.文档 DevOps 提供灵活的内容动态筛选和不同类型格式的定制导出.

(3) 用户体验:用户获取的信息不仅仅是静态的文档,还有音视频信息、模拟和互动式的指导信息.移动互联网的普及,也引出移动终端的访问等需求.文档 DevOps 提供多样化的发布形式及多样化的访问获取方式.

2.2.4 反馈环节

反馈环节是文档的评价、故障和建议的反馈的环节.这一环节一般内建在线上发布与浏览系统中,内部读者可以通过这个系统进行评审意见管理和问题跟踪,外部读者可以通过系统进行感受评价、问题反馈和提出建议.这些跟踪项将进行闭环管理,往往由提出人负责验收关闭.此环节有如下特性.

(1) 快速反馈:作者在完成并提交信息单元后,产品文档的构建和呈现在分钟级完成,审核人员或文档用户可以对文档进行在线浏览,并直接进行批注、评论、打分等质量反馈活动,快速反馈让编写人员、审核人员及用户的沟通顺畅、频繁.

(2) 快速闭环:用户反馈的问题快速传递到作者后,可以快速得以修复,文档修订后的结果也能快速发布给用户,形成闭环.快速闭环减少了错误最终出现的机会,文档质量得以提升,文档交付速度得到提升.

从上述各个环节的特点分析来看,文档 DevOps 方案较好地支持了敏捷研发模式下的大型软件系统的文档特性要求.

采用文档 DevOps 后,生产环节的全员协同和组件化+构建环节的自动化和可视化+反馈环节的快速闭环的特性,会对准确性提升有明显贡献,同样地,表 2 中还分析了文档完整性、及时性和可用性所涉及到的各个环节的关键特性.

Table 2 The effort of all activities of the solution

表 2 解决方案各环节的贡献表

环节	特性	准确性	完整性	及时性	可用性
生产	全流程		√	√	
	全员协同	√	√		√
	组件化标准化	√			
构建	代码化			√	
	自动化	√		√	
	可视化	√			√
发布	体系化		√		
	定制化				√
	用户体验				√
反馈	快速反馈			√	√
	快速闭环	√			√

2.3 系统架构

文档 DevOps 是涵盖文档全生命周期的工具集和完整的解决方案,整体架构如图 7 所示.

Fig.7 The architecture of document DevOps

图 7 文档 DevOps 系统架构

文档 DevOps 都建立在软件版本的 DevOps 工具链和云设施平台上,以确保和软件开发流程与文档开发流程的融合,便于软件研发和内容编写同步进行.

中间是文档 DevOps 平台,通过构建的产品文档的交付工具链和提供的服务,让文档编写人员专注于内容创作.

建立在文档 DevOps 平台之上的是面向产品文档解决方案,由各类 Pipeline 和具体配置构成,方便、快速地提供不同文档的开发和发布环境.

文档 DevOps 的 3 层模型间是有依赖关系的,上层的功能是下层工具链提供的.例如解决方案层的 FG/FD 文档生成流水线,由中层的文档 DevOps 平台的各种服务/工具组合而成;进一步地,中层平台中的文档 CI 服务,能够将代码化后的文档进行类似编译转换的过程,生成最终交付文档,这一过程又依赖版本 DevOps 工具链中的各类处理代码的工具支撑.

文档 DevOps 平台是文档 DevOps 整体解决方案的核心和主体,我们把对应的实现系统称为 iDoc,其主要包含 4 个部分,如图 8 所示.

(1) 领域模型库

领域模型库中包含了各领域的内容输出模版,比如需求模版、方案模版等等.代码、模型脚本或注释中包含的信息也可以作为文档素材来源,所以也是领域模型库的一个部分.每个模版中包含了不同的章节,每个章节信息类似 DITA 中的 Topic,是相对独立的标准化的信息单元,也定义了单元模版.iDOC 的模型库提供标准化的模版,各产品可以直接引用标准化的模版,也可以结合项目自身特殊情况进行适当的优化或定制本项目模版.

信息单元是对交付文档的信息架构进行分析,提炼相对独立的信息元素,再结合软件交付流程输入的需要,整理各环节的信息内容的输出,消除重复信息后,进行标准化设计后形成的.每个信息单元都有明确的角色和责任人,承接该单元内容输出.

(2) 内容生产工具

在敏捷的研发模式下,文档也是由全功能团队协同完成交付,不同岗位的角色在工具选择上往往存在差异,

为减少项目在写作习惯上的切换成本,iDOC 具备灵活支持多种不同生产工具编写的能力.比如 WIKI,提供了良好的多人在线协同编写的功能,有很多企业用于内外部文档协作[26];Word 是最为常见的文档编写工具,规划或工服岗位已经习惯并且经常有离线编写场景;MarkDown、ReStructureText 是开源相关的项目研发人员保持与社区同步的文档编写方式;XML 是技术写作者最熟悉和专业的协作方式.

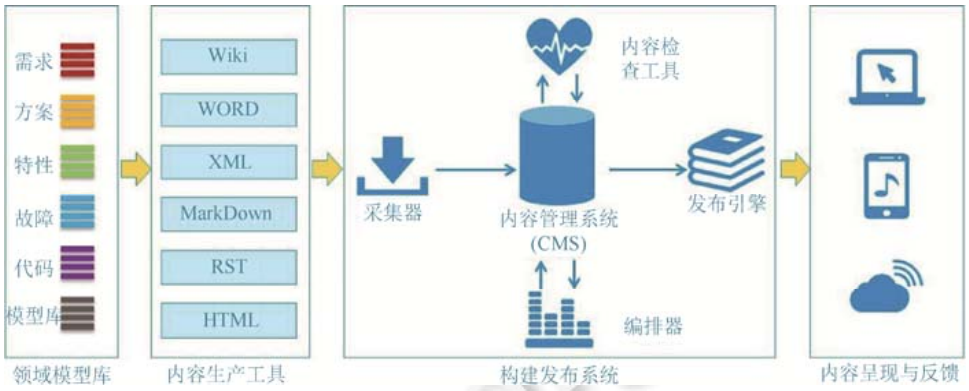

Fig.8 The platform framwork of document DevOps

图 8 文档 DevOps 平台架构

通过对多样化工具的支撑,产品的各类岗位都可以用最熟悉的方式输出信息单元和领域模版的内容.

(3) 构建发布系统

构建发布系统是 iDOC 的核心部分,在引用了 DevOps 的自动化工具,如 Jekins、Gerrit、制品库的基础上,开发了采集器、扫描器、内容管理系统、编排器、发布引擎 5 个大子系统.

• 采集器:根据项目的配置信息,负责对多种格式的信息单元的来源进行自动化的采集、转换以及内容的标准化.

• 扫描器:根据通用检查规则及项目的自定义规则,负责对信息单元内容的自动化扫描检测,及时反馈内容编写的错误,如用词规范、单词拼写、模版规范性等问题.

• 内容管理系统(content mangament system,简称 CMS):CMS 对项目的所有内容素材进行代码化的管理,可以基于内容进行版本分支管理,实现文档信息的历史版本回溯与修订.

• 编排器:根据发布文档类型,编排器按照不同的编排规则进行信息单元的自动化的组合拼装.项目可以直接引用 iDOC 的通用编排规则,也可以通过配置脚本进行编排方式的调整.

• 发布引擎:根据编排器生成的内容信息,发布引擎生成各种类型的外发格式的文档,如自定义文档包、PDF、WORD、CHM 等等.同时还可以根据用户需要,动态定制内容,生成个性化的文档包.

(4) 内容呈现与反馈

文档发布之后,iDOC 提供多样化的文档浏览获取方式,如在线的技术支持网站、离线阅读器、移动终端访问阅读等.同时,在各种浏览方式下,提供快速发批注、评论、打分等功能,并将相关信息传递给作者,作者可以进一步交互互动,对文档进行修订,形成快速反馈和闭环.

3 文档 DevOps 方案实现

为了能够有效地解决大型系统产品文档交付的“四性”:准确性、完整性、及时性和可用性,文档 DevOps 必须具备以下属性.

(1) 系统软件的产品文档都是由多人协同才能完成的,同时,文档与文档之间的包含引用关系也非常复杂,为了减少人为的抛接和内容的重复编写,降低文档的维护成本,提高文档的准确性和完整性,要求文档 DevOps 能够支持复杂文档信息架构和引用关系可配置,同时支持内容做到一次编写,多次引用,做到同源维护.

(2) 系统软件一般由多种组件组合集成发布,比如开源组件、自研组件等,产品文档也是由相关组件的文档

进行集成而发布的.而每种组件的文档开发方式可能存在不一致,比如开源采用 MarkDown 方式,自研组件可能采用 Wiki 或 DITA 的方式进行开发,但是文档最后发布需要统一文档包发布,这就要求文档 DevOps 必须要能支持多种开发格式,最后统一格式打包发布,因此,文档 DevOps 需要有多种格式转换的统一标准和平台,以便支持文档信息架构和引用关系的编排.

(3) 让文档写作者在研发过程中只关注于文档内容的编写,其他交付过程全部自动化,从而提高交付效率,做到文档和版本同步发布,解决文档交付的及时性.因此,要求文档 DevOps 的 CI 效率必须对齐软件 DevOps.基本前提就是要实现文档代码化管理,基于代码管理的易分析、可自动检查、可构建等特性实现 CI 效率提效.

3.1 内容同源可灵活配置

实现内容同源可灵活配置,需要具备如下 3 个关键要素,其中,前两个要素是内容同源的基础,是第 3 个要素的前提.

(1) 文档架构设计,根据文档的用户需求,优化交付文档的结构.

结构化的文档是内容同源的基础.项目针对用户的工作场景和工作任务进行分析调研,结合用户需求,由文档架构师对现有文档体系、各类文档的章节进行优化,确保每篇文档内容是用户所需要的,每篇文档内容完整,其中每个章节的内容相对独立和完整.

(2) 信息单元梳理,信息单元的承接人员、流程活动、输出样板.

外部文档的架构确定之后,需要梳理文档相关的信息单元.

• 根据外部文档分析梳理信息单元.根据外部文档的重要性排序,由文档架构师与项目的系统工程师、业务分析师 BA(business analysist)研讨,逐一分析文档中各章节的内容,将章节内容拆分为各领域的信息单元,类似 DITA 中的 Topic.信息单元不能过小,需要具备信息描述的完备性,但也不宜过粗,以免影响信息单元的可重用性.例如,特性指导书,应该包含特性描述、特性背景、特性实现原理、特性配置等等内容.分析过程中,需要记录文档章节与信息单元的关系,如拼装、链接、引用等,记录信息单元内容要求.

• 根据软件交付分析梳理信息单元.根据软件活动的输入和输出,由 EPG 与领域专家分析整个产品的交付流程中每个环节和活动应该输出的信息单元.例如,开发实现活动中,需要有特性的实现原理、特性的交互、特性的关键技术等信息单元.分析过程中,需要记录流程活动与信息单元的关系.

• 整个文档交付体系、软件交付流程梳理完成之后,对分解出大量的信息单元进行领域的归类,对各领域的信息单元进行审核,内容有重复交叉的信息单元,要消除重复或进行信息单元的拆分;对于信息单元包含内容过多的要拆分,颗粒度以单个角色在单个领域活动中可独立完成为适宜.信息单元优化和调整后,要同步修订文档与变更之后的信息单元之间的关系.最终可以在各个领域得到软件的完整信息单元,例如用户需求领域的信息单元、特性领域的信息单元...各领域的信息单元可以共同来支撑整个文档体系的交付与软件的交付.

由图 9 所示的示意图可以看到,外部交付和产品交付的内容与过程内容之间存在着同源关系.

Fig.9 The diagram of content single sourcing

图 9 内容同源示意图

• 与各领域的代表商定承担信息单元编写的领域责任人及编写要求.根据领域的信息单元之间的关联关系,可以确定领域的模型模版,如用户需求说明书模版、特性说明书模版,其中每个章节都属于某个信息单元,有对应的责任人和编写样例.

图 10 所示为经过信息单元分解之后,各流程活动中对应角色需要输出的需求领域的信息单元,这些信息单元组成了需求说明书、特性说明书的模版,最终,其中部分信息单元会抽取到交付文档特性指导书中.

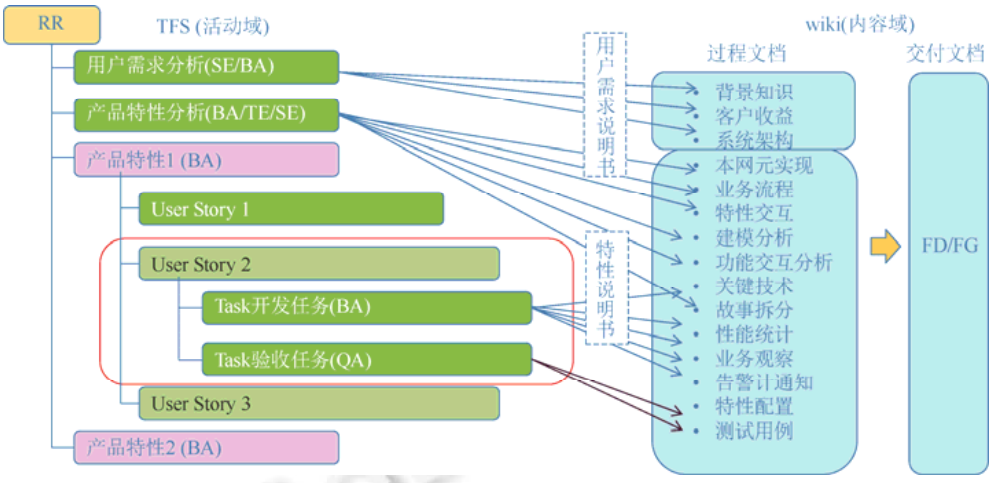

Fig.10 The association of R&D process and document content

图 10 研发过程和文档内容关联示意图

(3) 灵活配置管理,利用编排器组装生成最终文档.

上一步骤中记录的文档与信息单元之间的关联关系,可以提取出交付文档与信息单元之间的配置关系,形成文档 Devops 编排器的配置文件.编排器结合配置文件,可以实现各类信息单元的自动化组合拼装.配置文件纳入内容管理系统中,基线化管理,一旦配置好之后,可以持续运行,并且可以随着文档架构的优化,动态调整配置信息.

3.2 多格式转换的标准和平台

如系统架构的生产工具所示,文档 DevOps 需要面对多样化的编辑工具和多种内容格式的信息单元,最终的文档可能是来自不同格式的组合拼装而成的,需要解决如下问题.

(1) 采集信息单元的标准化

采集器的主要功能是原始信息单元的采集,另外还有一个重要的功能就是信息单元标准化.即将多种格式统一转换到 HTML,并且对信息单元进行识别和标准化,这个过程给后续的内容检查、编排、发布等环节提供了统一格式的信息单元,降低了整个系统的处理复杂度.如果有项目在生产工具上有迁移,只需调整采集器的配置即可;有新格式的工具引入,也只需要增加采集器对新格式的适配,项目即可完整使用 iDOC 系统的原有的一系列功能,如内容检查、编排、多种文档发布等等.

(2) 统一的内容管理系统

MarkDown、ReStructureText、XML、HTML 等格式与代码接近,适合于用 SVN 或 Git 来管理,但 Wiki 是自己有一套数据库来管理内容,因此给内容的统一管理形成了障碍.而且,Wiki 的内容管理是基于单页面的,每个页面都有相对独立的版本号来管理,无法进行多页面内容的统一版本管理,无法满足产品的批量文档内容的版本管理需求.

iDOC 的 CMS 具备管理 Wiki 的在线系统内容的能力.通过 Jekins 系统调度周期性的检测工具,检测 Wiki 空间中各个页面的更新情况,一旦有内容修改,触发采集器对内容进行提取,保存在 Git 中,确保内容的完整性和

版本管理的一致性.

并且,iDOC 在 Git 的分支管理基础上增加 Wiki 的分支内容的检出功能,可以根据需要将 Git 中历史分支内容检出到 Wiki,内容修订完成后可以采集修订后的内容到 Git 中,实现对历史版本的文档内容的变更.

3.3 文档CI的效率

文档 DevOps 要让文档编写人员只要专注于内容编写,就需要解决如下几个问题.

即见即所得:文档开发人员在编辑环境编写的内容要能在发布环境快速看到最终结果,便于快速反馈文档编写质量.因此要求从文档开发到文档发布的流水线效率必须在分钟级的效率范围内.

文档质量内建:文档格式、文字规范性、内容正确性和规范性要求等有明确规则和规范要求的都需要在流水线中自动检查,及时反馈并修订,避免低级或规范性的错误,提高文档质量.

解决以上两个问题的核心要素就是文档代码化管理.为了支持代码化管理,需要支持标记性语言存储,标记性语言便于分析,比对分析和自动化检查,同时可支持灵活组合和打包,因此,文档 DevOps 可支持 HTML、XML、RST 等格式的存储和管理.

代码化编写文档后,需要将文档从编写格式转换成发布格式的过程,我们称其为发布过程.例如:通过 DITA 进行 XML 源文件到 HTML/PDF 等格式在编译转换过程.

发布过程是一个高频和大范围使用的功能,编译提速的意义非常大.使用 iDoc 能将单次发布速度由原来的 3 小时缩短到 2~5 分钟(如图 11 所示),原本发布周期也由原来的几天一次,变为现在可以随时获取到整合好的发布版本.全流程周期缩短了 96%,人工工作占用大为降低.

	SCM checkout	Tools checkout	Tools build	Wiki download	Wiki format	SCM commit
Average stage times:	2min 49s	4s	18s	24s	9s	58s
#308 Feb 27 12.06 No Changes	2min 33s	3s	18s	37s	9s	46s
#307 Feb 27 00.06 No Changes	2min 34s	3s	13s	5s	9s	55s
#306 Feb 26 12.06 No Changes	2min 38s	4s	22s	5s	10s	39s
#305 Feb 26 00.06 No Changes	2min 31s	3s	10s	7s	9s	49s
#304 Feb 25 12.06 No Changes	2min 36s	3s	22s	24s	9s	45s

Fig.11 The CI of document

图 11 文档 CI

3.4 方案实施实例

举例说明:图 10 中 FG 文档的内容素材主要来自于两篇过程文档《用户需求说明书》和《特性说明书》.

(1) 在生产环节中,通过“FG 文档架构工作坊”的活动,完成 FG 文档架构规划和同源信息单元的梳理,明确《用户需求说明书》和《特性说明书》各个信息单元的开发能够满足 FG 文档的交付.接下来,对各个信息单元的开发活动进行跟踪管理,描述如下.

- 背景知识、客户收益、系统架构,来源于需求活动之用户需求环节,我们的市场专家输出;
- 特性分析、关键技术、业务流程等,由 BA 在产品需求环节,进一步补充;

• 特性配置、测试用例等,由 QA 在测试验证环节,完善素材;

• 其余的参数、计数器、告警等,散落在研发的各个环节中进行输出.

(2) 在构建环节,利用“FG 文档架构工作坊”的梳理结果,生成配置文件表达 FG 文档的架构设计,将素材的关系、逻辑结构进行准确定义.当生产环节的信息单元被修改后,文档 DevOps 工具链自动地将各个信息单元部分连接成最终交付文档,其中一个配置文件如图 12 所示.

```
<?xml version="1.0"?>
<condition type="h2" value="功能简介" note="">
    <content type="h1" class="node" value="范围" note=""/>
    <content type="h1" class="node" value="概述" note="">
        <content type="h2"  value="功能简介" note=""/>
        <content type="h2"  value="许可控制" note=""/>
        <content type="h2"  value="关联特性" note=""/>
        <content type="h2"  value="关联服务" note=""/>
        <content type="h2"  value="变更说明" note=""/>
        <content type="h2"  value="组网方式" note=""/>
    </content>
    <content type="h1" class="node" value="技术描述" note="">
        <content type="h2"  value="术语和定义" note=""/>
        <content type="h2"  value="技术原理" note=""/>
    </content>
    <content type="h1"  class="node" value="开通指导" note=""/>
    <content type="h1"  class="node" value="关联的计数器、KPI统计及告警" note="">
        <content type="h2" class="node" value="关联的计数器" note=""/>
        <content type="h2" class="node" value="关联的KPI统计" note=""/>
        <content type="h2" class="node" value="关联的告警" note=""/>
    </content>
    <content type="h1"  class="node" value="网络影响" note=""/>
    <content type="h1"  class="node" value="功能测试" note=""/>
    <content type="h1"  class="node" value="缩略语" note=""/>
    <content type="h1"  class="node" value="参考文档" note=""/>
</condition>
```

Fig.12 The configure file

图 12 某配置文件

工具链根据配置文件完成内容编排、内容生成、内容格式化、内容检查、内容打包输出等过程后,编译出文档后就进入到下一个环节.

(3) 在发布环节,文档 DevOps 平台自动地将内容部署到内部在线系统上进行阅读评审,结束后根据版本发布决策,阶段性部署到公司的制品库上,供工服、客户进行下载使用,也可以部署到 e 读服务器进行离线阅读.

(4) 反馈环节,会包含在我们内外部的在线系统中,包括两个部分.

(a) 针对研发过程的内部反馈,包括内容检查结果、批注建议,形成 FG 阅读质量和体验的度量数据,便于持续改进;

(b) 针对交付之后的外部反馈,包括客户满意度、客户意见等,利用故障跟踪管理系统,对 FG 文档进一步改进提升和问题闭环.

4 实施效果

从 ZTE 开始推广实施敏捷开始,一部分研发项目就意识到敏捷研发模式下的文档交付的问题,开始有项目探索文档的敏捷交付实践,进行文档持续集成的尝试和文档自动化检查工具的应用;内容建设上,由文档架构师、项目技术骨干、技术教练合作,进行同源的规划设计,试点项目的效率和质量上有明显提升;2017 年正式提出文档 DevOps 的概念,并将其作为基础设施数字化建设的 3 条主线之一,开始加快平台 iDoc 的建设和在公司内推广使用.

截至 2018 年 6 月,文档 DevOps 平台已支持 60 多个项目.实施项目以产品研发项目为主,产品类型覆盖有线系统产品、无线系统产品及终端产品.研发项目的规模,从 20 左右到 1 000 人以上的项目都有,其中一半左右

的项目规模在 50~300 人之间.

已有结果度量数据的约 23 个项目中,为观察典型项目的收益,我们选取了有代表性的 9 个项目作为分析样本,其规模和产出文档数见表 3.

Table 3 The typical projects

表 3 选取的典型项目

项目名	人数	篇数
a	78	99
b	132	96
c	110	128
d	123	140
e	96	88
n(新项目)	260	30
g	88	78
s	126	154
u	105	78

偏大和偏小的项目,在文档 DevOps 实施后的优化效果上,也有不同程度的体现.但本文研究对象主要考虑系统产品类的典型项目,目的是确保我们的改进对这些项目是有效的,对其他类型项目的有效性缺乏更多数据,并不是本文重点.

项目基于文档 DevOps 平台的交付文档类型,包括售前规划主导交付的特性说明、售后主导交付的版本发布说明书、研发主导交付的特性指导书、在线帮助、参数手册等文档,现阶段收益比较明显的是特性指导书、版本发布说明书和在线帮助.

从实施后的度量数据看,在 4 个方面有明显的改进效果.

(1) 周期缩短,成本降低

表 4 是某类系统产品项目中新编写单篇特性指导书的效果数据.平均投入时长整体的时间周期缩短了 30%以上,人力投入减少 44%以上.

Table 4 The effort of feature guides delivery

表 4 特性指导书交付效果

特性指导书	生产环节			构建发布环节		
	使用前	使用后	效果	使用前	使用后	效果
时长(小时)	58	40	缩短 31%	1	0.04	缩短 96%
人力投入(人时)	64	36	减少 44%	1	0	不需要人力

特性指导书原来是抛接方式:研发人员写完过程中的需求特性文档,再编写交付文档素材,技术写作者再重新写交付文档;文档 DevOps 实施后,3 个类型的文档通过信息单元精简合一,研发人员与技术写作者在过程中协同编写的信息单元,重复内容的编写消除了,文档的周期缩短,投入也有所降低.

表 5 是某类系统产品项目中单篇版本发布说明书的效果数据.一个中等规模的发布版本所涉及的需求条目、故障条目分别在 30 条左右,其人力投入由原来 4 人天左右减少到 1 人天,投入大幅缩减 75%以上,文档准确性得到提升.

Table 5 The effort of release notes delivery

表 5 版本发布说明书交付效果

版本发布说明书	生产环节			构建发布环节		
	使用前	使用后	效果	使用前	使用后	效果
人力投入(人时)	32	8	减少 75%	1	0	不需要人力

版本发布说明书原来纯人工收集各类版本相关的需求、故障信息,再汇总整理成文档,耗时耗力,也容易出错;实施文档 DevOps,可以自动化地采集各需求管理、故障管理系统的信息.

在线帮助内容原来由研发人员编写,与技术写作者的其他用户文档有很大部分内容存在重复.某类系统产品项目实施文档 DevOps 之后,研发人员与技术写作者协同编写一份内容,同步生成帮助内容和用户手册两类文档,降低人力成本 40%左右.

(2) 及时性提高

文档 DevOps 有流程融合的特点,在软件交付的过程中,文档的信息单元同步输出,文档内容在软件交付过程中迭代生成和补充完善,文档与软件同步交付.

根据某类系统产品项目实施文档 DevOps 一年多来的文档发布及时性统计数据,特性指导书等交付文档滞后版本时长从实施之前的平均 2 个月以上,缩短到平均 2 天左右.

(3) 质量显著提高

通过文档 DevOps 的实施,确保文档的信息单元是研发流程中某些重要环节的输入或输出,如特性说明书是软件特性实现的输入,参数说明是软件的领域模型的输出.信息单元与软件开发之间的密切关系,促进了文档与软件之间的一致性、准确性、完整性明显增强.某类系统产品项目对特性指导书、参数手册等文档近 1 年的故障数据统计分析,上述 3 类问题下降 70%以上.

DevOps 的基本理念是嵌入质量保证手段,特别是使用实时的用户监控发现问题[27].文档 DevOps 遵循这个理念,提供了 CI 自动化快速反馈,提供实时的浏览和批注等功能,质量改进效果也比较明显.原本需要人工检查的问题,应用了自动化检查工具后,针对中文、英文的文档提交过程中尽早地发现问题.某类系统产品项目版本发布闭环的故障均超过千次以上,文档质量问题的闭环平均周期缩短 60%以上.

(4) 可用性有待数据验证

对于文档的可用性,当前内部用户反馈效果较好,暂时没有量化结果,还有待进一步收集内外部数据.

5 讨 论

5.1 改善的原因分析

取得文档过程的改善,我们认为主要的原因有:

(1) 做到文档开发过程和软件敏捷开发过程融合,使得文档的内容由最熟悉的人在开发过程中同步输出.研究表明,DevOps 高效流程可以提效超过 20%以上[28],原来的模式中,两个过程是分离且串行的,导致大量的信息传递;现在文档 DevOps 的做法提升了文档的及时性、准确性和可用性.

(2) 通过信息流梳理,对研发过程需要输出的信息单元进行了分析和解耦,利于第 1 条的同时,确保完整涵盖最后产品文档的交付要求.原来的模式中,缺乏一个明确的过程内容和交付文档间的对应关系管理.现在的做法提升了文档的完整性、准确性和可用性.

(3) 在内容的编写过程中,通过文档 CI 实时地给予质量反馈.这样的反馈,是面向最终要交付的产品文档,即让所有人在输出内容时都遵循交付的标准.原来的文档内容的反馈周期长,反馈时间点滞后.现在的做法提升了文档的及时性和准确性.

本文解决的问题,在业界也有一定的典型性,可以为 DevOps 实施企业提供参考.

5.2 适用性与限制

使用文档 DevOps 的方式,对研发过程中的工程师,提出了文档写作上的要求.

(1) 站在用户的角度编写内容

研发工程师描述内容时,很容易站在实现者的立场,而不是用户的角度,这个需要意识上的持续转变.

(2) 需要更加细致的描述

研发人员常常讨论完内容后,并不详细描述相关的文档,而是直接开始编写代码和用例,这会导致本来已经显性化的知识重新变为隐形,对知识传播非常不利.使用了文档 DevOps 后,会有利于改变这种情况,但需要研发人员对讨论的结果进行细致的整理和文字描述.

(3) 样式标签与内容解耦和标准化

大量的信息不再通过抛接给技术写作者来编写,而是由研发人员在开发过程中整理好,这就需要支撑的系统能够很好地将样式标签与内容解耦和标准化,并对一些工具限制对研发人员进行培训.

文档 DevOps 的流程与架构适用已经有敏捷和 DevOps 基础的团队,对于软件版本的 DevOps 的运作成熟度,会直接影响到文档 DevOps 的执行效果,原因是文档 DevOps 的很多流程是内嵌到软件版本的研发流程中的.

另一个限制性来自于文档 DevOps 面对的产品文档的变化程度.实施文档 DevOps 的过程,将涉及到大量内外部文档内容与格式的分析与定义,需要的相关干系人共同参与演进过程,这会消耗大量的人力物力,对于已经成熟的产品,其产品文档变化已经不会很大,完整实施文档 DevOps 的性价比低,不适用去改造.

5.3 文档价值交付的文化转变

区别于传统文档交付,在文档 DevOps 的交付方式中,文档内容编写由技术写作者主导转移到研发人员同步编写.对于研发人员而言,看似额外多出一项非开发任务,而且疑似和敏捷价值观中的"可用的软件胜于面面俱到的文档"有所冲突,转型初期可能会出现研发人员抵触文档写作的现象.

实际上,敏捷价值观中简化文档的本质是指要消除不必要的文档,而本文中的文档 DevOps 中涉及到的文档都是需要对外交付的必要文档,是产品价值的重要部分,因此不但没有冲突,而且如第 3.1 节中的同源引用的方法有助于团队判断文档的必要性,从而可以消除多余的文档.

另外,为了确保研发人员有意愿和有能力承接文档开发工作,对组织文化也要有如下转变.

(1) 引导职责共享文化:敏捷倡导全功能团队,聚焦价值交付.而文档作为产品价值交付范围之一,团队也需为文档交付负责.文档 DevOps 中的协同开发可以支持团队共同完成软件和文档的同步交付,做到职责共享.

(2) 建立价值交付导向的业绩评价标准:传统研发模式中评价研发人员业绩仅仅关注开发阶段的工作业绩,而忽略整体价值交付评价.在敏捷中强化端到端的价值交付理念,评价个人业绩也转向其在价值交付中的贡献.这样使得团队和个人愿意认领文档开发的任务.

(3) 打造专注和极致的研发环境:文档 DevOps 通过简化文档开发复杂度,让研发人员专注于内容开发.其中,文档代码化管理方式使得研发人员的工作方式类同于软件开发,通过编排和模板屏蔽了文档格式要求,降低了文档开发的技术门槛,研发人员只要专注于他熟悉的内容表达,可一次性高质量完成文档交付.

5.4 固有缺陷说明

文档 DevOps 的关键点是关于文档内容的同源配置,这是它发挥价值的重要环节,也是其难点和固有缺陷的来源.

(1) 同源配置来自于对文档应用领域的准确理解和划分,但这个过程是一个渐进的过程,所以会要求配置环节比较灵活,对稳定性和控制复杂度要求都很高.

(2) 不同用户对相同领域知识的定义,可能产生冲突,特别是在商业模式创新的领域,这将导致内容同源部分还是会必然引入重复或冗余,无法完全消除.但这个过程的分寸如何拿捏将会非常考验信息架构师的能力.

5.5 对研发人员开发效率的影响

敏捷文档交付的一个重要转变就是让开发人员直接参与最终交付文档的创作,这看似会占用开发人员的本职开发工作时间,但也有可能会影响到原有的开发效率.

事实上,开发人员在传统的开发过程中,为了确保开发质量,通常会编写过程文档,比如方案设计、详细设计等文档用于评审,这一般会占用开发 10%~20%的时间.而且为了配合文档创作者交付文档,开发人员还需要为文档创作者提供额外的素材文档,这个也需要占用开发时间.敏捷文档交付之后,开发人员在开发过程中就按照对外文档的要求输出文档内容,该文档可用于内部过程质量评审,同时还满足对外交付要求.相对于开发人员的时间而言,还节省了对外文档素材整理和交流的时间,所以,从开发人员投入总体开发时间来看,不会有太大的影响.

据某系统产品的单个需求开发人力成本统计,敏捷转型初期需求人力开发成本呈下降趋势,敏捷转型之后需求人力成本基本保持稳定,说明文档 DevOps 方式的应用并没有影响到开发本身的效率.如图 13 所示.

Fig.13 The labor cost per requirement of a project

图 13 某项目每需求人力成本统计

5.6 未来工作

5.6.1 社区化

文档更易于被用户分享和传播,带来更多阅读;满足用户更多的需求,用户更便于表达建设性的反馈;文档成为社区化交流的媒介.

每一个阶段的社区化可以为文档编写发挥的作用,如图 14 所示。

Fig.14 The revenue of community-based approach of document DevOps

图 14 文档 DevOps 社区化方式收益分析

5.6.2 云 化

云计算支撑了文档生产、获取、消费过程全部云化.文档云化托管,从云端按需存取.

整个研发模式,需要建立在云技术基础上,这样才能真正发挥文档 DevOps 的快速协作的威力,满足跨产品的同源要求.

5.6.3 智能化

人工智能作用在人、知识、产品这 3 个领域间快速建立链接的行为上,是大势所趋.首先在文档内容生产、消费中,AI 已经崭露头角.

(1) 给出同步翻译的辅助指导

大规模软件系统往往是国际化产品,研发人员一般只会使用母语写作其中的内容,其他语言版本都需要专业人员翻译,如果能够在翻译的同时,给出高质量的机器翻译作为辅助参考,那么会极大地提升专业翻译的效率.文档 DevOps 中能够集成同步翻译的辅助指导,是一个重要的改进.

(2) 人工智能进行可读性辅助判断

可读性是一个模糊领域的问题,可以通过分析技术写作者持续地修改记录,利用人工智能技术帮助提示出一些可读性改进的建议,甚至将来 80%的可读性问题主要由工具来指出和完善.

6 总 结

专业咨询机构的调查显示,DevOps 将会在系统软件研发中得以大规模应用.本文重点讨论的内容有:完善 DevOps 在文档领域的交付模式,探索其系统架构,并解决其中的关键问题.本文把这些工作统称为文档 DevOps.

文档 DevOps 主要解决面向系统软件的软件版本和产品文档同步交付的挑战.在敏捷和 DevOps 变革后,软件版本发布节奏变得更加频繁,而与之配套的文档交付模式没有发生对应变化时,导致了系统软件的敏捷价值交付无法实现.

本文提供了文档开发相关的流程与架构实现,通过如下 3 个方面的工作来让软件版本和产品文档同步发布,并且利用 DevOps 的快速反馈优势提升了文档开发协作效率和产品文档的质量.

(1) 对信息流上的内容进行架构解耦,再利用工具/平台进行同源引用,组合生成需要的各类文档,增加了过程文档与交付文档、交付文档之间的内容复用,提升了文档准确性,同时,分析和改进产品文档的完整性.

(2) 构造文档的 CI 流水线,并在流水线中内嵌自动化质量检查工具,做到快速反馈和质量内建,守护产品文档的准确性和完整性.

(3) 将文档的开发过程与软件版本的 DevOps 融合,在统一的版本研发流程中,进行文档的版本管理,确保产品文档与软件版本的同步开发交付,并打通用户的反馈.

文档 DevOps 在几十个项目中已被实际应用,也获得了初步的推广效果.进一步地,本文工作的意义还在于扩充了 DevOps 的适用范围,涵盖了版本和文档两类对象.对采用敏捷开发模式的项目,文档 DevOps 的补充也完整地支持了项目的全部主要工作流程,为系统产品的价值交付奠定了基础.

References:

[1] Janessa R, Van der Rob M. Gartner says by 2016, devops will evolve from a niche to a mainstream strategy employed by 25 percent of global 2000 organizations, 2015. http://www.gartner.com/newsroom/id/2999017

[2] RightScale. 2016 state of the cloud report. 2016. https://www.rightscale.com/lp/devops-trends-report

[3] RightScale. 2017 state of the cloud report. 2017. http://www.rightscale.com.cn/pdf/RightScale-2017-State-of-the-Cloud-Report.pdf

[4] Van Heesch U, Theunissen T, Zimmermann O, Zdun U. Software specification and documentation in continuous software development: A focus group report. In: Proc. of the 22nd European Conf. on Pattern Languages of Programs. 2017. Article No.35.

[5] van Heesch U, *et al*. Software specification in continuous software development—A focus group report. Report, EuroPLoP, 2017. 13.

[6] Merilinna J, Matinlassi M. State of the art and practice of open source component integration. In: Proc. of the 32nd EUROMICRO Conf. on Software Engineering and Advanced Applications, EUROMICRO 2006. Washington: IEEE, 2006. 170–177.

[7] Sommerville I. Software documentation process. In: Thayer RH, Dorfman M, eds. Software Engineering, Vol 2: The Supporting Processes. 3rd ed., Hoboken: Wiley-IEEE Press, 2002. 141–164.

[8] Briand LC. Software documentation: How much is enough. In: Proc. of the 7th European Conf. on Software Maintenance and Reengineering, CSMR 2003. Washington: IEEE, 2003. 13–15.

[9] de Souza SCB, Anquetil N, de Oliveira KM. 2005. A study of the documentation essential to software maintenance. In: Proc. of the 23rd Annual Int'l Conf. on Design of Communication: Documenting & Designing for Pervasive Information, SIGDOC 2005. New York: ACM, 2005. 68–75.

[10] Forward A, Lethbridge TC. The relevance of software documentation, tools and technologies: A survey. In: Proc. of the 2002 ACM Symp. on Document Engineering, DocEng 2002. New York: ACM, 2002. 26–33.

[11] Dorairaj S, Noble J, Malik P. Knowledge management in distributed agile software development. In: Proc. of the Agile Conf. (AGILE). IEEE, 2012. 64–73.

[12] Hadar I, Sherman S, Hadar E, Harrison JJ. Less is more: Architecture documentation for agile development. In: Proc. of the 6th Int'l Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). 2013. 121–124.

[13] The Agile Manifesto. http://agilemanifesto.org/

[14] Dingsoyr T, Nerur S, Balijepally V, Moe NB. A decade of agile methodologies. Towards explaining agile software development. Journal of Systems and Software, 2012,85(6):1213–1221.

[15] Prause CR, Durdik Z. Architectural design and documentation: Waste in agile development. In: Proc. of the Int'l Conf. on Software and System Process. Zurich, 2012. 130–134.

[16] Tang A, Gerrits T, Nacken P, van Vliet H. On the responsibilities of software architects and software engineers in an agile environment: Who should do what. In: Proc. of the 4th Int'l Workshop on Social Software Engineering (SSE). Szeged, 2011. 11–18.

[17] Priestley M, Hargis G, Carpenter S. DITA: An XMLbased technical documentation authoring and publishing architecture. Technical Communication, 2001,48(3):352–367.

[18] Priestley M. DITA XML: A reuse by reference architecture for technical documentation. In: Proc. of the 19th Annual Int'l Conf. on Computer Documentation. 2001. [doi: 10.1145/501516.501547]

[19] Organization for the Advancement of Structured Information Standards. Darwin Information Typing Architecture (DITA) Version 1.3 Part 2: Technical Content Edition. OASIS Standard. 2016. http://docs.oasis-open.org/dita/dita/v1.3/dita-v1.3-part2-tech-content.html

[20] Kaplan N. The death of technical writing, part 1. 2014. https://customersandcontent.com/2014/05/03/the-death-of-technical-writing-part-1

[21] Evia C. Authoring standards-based intelligent content te easy way with lightweight DITA. In: Proc. of the SIGDOC 2017. 2017. 5.

[22] Díaz O, Anfurrutia FI, Kortabitarte J. Using DITA for documenting software product lines. In: Proc. of the 9th ACM Symp. on Document Engineering. 2009. [doi: 10.1145/1600193.1600244]

[23] Gentle A. Continuous integration and delivery for documentation. 2015. https://opensource.com/business/15/7/continuous-integration-and-continuous-delivery-documentation

[24] McCance S. 5 trends in open source documentation. 2016. https://opensource.com/article/16/12/yearbook-5-trends-open-source-documentation

[25] Fraley L. Beyond theory: Making single-sourcing actually work. In: Proc. of the 21st Annual Int'l Conf. on Documentation, SIGDOC 2003. 2003. 52–59.

[26] Aguiar A, David G. WikiWiki weaving heterogeneous software artifacts. In: Proc. of the 2005 Int'l Symp. on Wikis. 2005. 67–74.

[27] Liu BH. 2018 state of DevOps in China report. DevOps Community of China, 2018 (in Chinese).

[28] Puppet. 2016 State of DevOps Report. 2016. https://puppet.com/resources/whitepaper/2016-state-of-devops-report

附中文参考文献:

[27] 刘博涵.2018 年中国 DevOps 发展报告.DevOps 中国社区,2018.

金泽锋(1977－),男,江西南昌人,学士,CCF 专业会员,主要研究领域为 DevOps,软件工程,敏捷开发.

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工程.

张佑文(1971－),男,工程师,主要研究领域为 DevOps,软件过程改进,敏捷开发.

邵栋(1976－),男,副教授,CCF 专业会员,主要研究领域为软件工程.

叶文华(1976－),女,硕士,主要研究领域为 DevOps,软件工程,敏捷开发.