S-SDLC CMM 软件安全开发能力成熟度模型

S - S D L C

文档修订记录

版本	变化状态	修订说明	日期	变更人	备注
V1.0	N	新建文档	2022-12-30	S-SDLC CMM	
V2.2	M	修订	2023-03-06	S-SDLC CMM	
V2.3	M	修订	2023-03-22	S-SDLC CMM	

目　录

Contents

作　　者

徐瑞祝、罗欣然、许昌恒、汪东星、莫逢涛、林文龙、车逸轩、吴宇莫

Part 01

模型概述

1. 背景

在过去十年的时间里，我们对金融、通讯、软件服务、房地产等行业的部分头部企业进行了软件安全开发体系建设的深入调研，结果显示很多软件开发组织或多或少执行了软件安全开发的实践，但他们其实并不清楚组织自身的安全开发能力处于哪一个水平，与行业最佳实践有哪些差距。虽然现有公开的一些安全开发能力评估模型都包含了常见的安全活动，但在执行细则上还是缺少相关标准，无法作为落地性的指导。企业如果缺乏整体性的规划，或者方向错误，就会导致安全建设的投入产出比低下，对企业的正常业务发展造成影响。

在此背景下，本项目致力于编写出范围更全面、落地指导性更强、更具适用性的安全开发能力评估框架模型，从而使企业利用该模型更加客观地认识自身软件安全开发的能力，并横向对比业界最佳实践，为软件开发组织提升软件安全开发能力提供依据。

2. 模型概述

2.1. 什么是S-SDLC CMM

S-SDLC CMM(S-SDLC Capability Maturity Model)是一个“指导式”的软件安全开发能力成熟度评估模型，可作为软件安全能力成熟度的量化评估依据和软件安全开发体系建设的参考指导。

S-SDLC CMM模型具有以下特性：

- 以观察打分为基础的评价体系，能更为客观地衡量企业软件安全水平，并给出针对性建议。
- 每年更新迭代国内外软件安全法规政策要求，跟进最新安全热点问题。
- 适用于瀑布式、敏捷式等多种软件开发模式。

2.2. S-SDLC CMM术语表

S-SDLC CMM：软件安全开发生命周期成熟度模型。

S-SDLC域：软件安全开发生命周期成熟度模型共分为4大评估域，分别为：监管域、能力域、触点域、运维域。

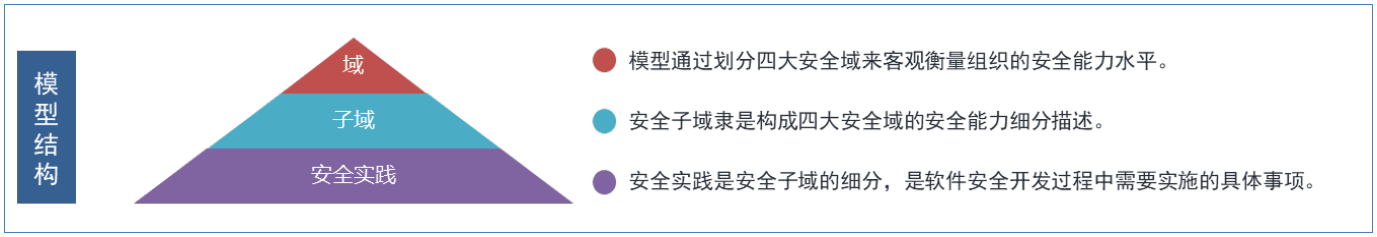
S-SDLC子域：4大评估域安全能力拆解为特定的能力域集合。

实践：需要企业执行的安全活动以及细则指导。

2.3. S-SDLC CMM结构

S-SDLC CMM模型包含了监管、能力、触点和运维四个大域，又细分为17个子域。模型通过划分安全域、安全子域、安全实践三层结构，将安全能力细化至可客观量化的执行动作，由此得出安全实践的执行水平，完成对组织安全能力成熟度的客观评估。

S-SDLC CMM对于四个大域的定义是：监管域描述了软件开发组织制定软件安全开发流程、合规要求、管理制度等的体系化文件；能力域描述了软件开发组织保障各种安全活动执行所具备的基础设施；触点域描述了软件开发组织在软件开发生命周期中应该如何融入安全实践；运维域描述了软件开发组织的生产环境和软件上线需要执行的安全实践。



S-SDLC CMM模型定义的成熟度是对17个子域综合评分得来的，最终的评分将在模型框架下进行同行最佳实践比较。

为指导软件开发组织实施S-SDLC，我们在子域中给出了具体的实践内容以及对应的参考指标，这些措施提炼自大量的实践经验以及业界的先进共识。组织通过实施模型提供的实践内容和参考指标进行改进，能切实提高软件安全开发的能力。S-SDLC CMM整体结构如图1。

S-SDLC CMM 4 大评估域及下属 17 个评估子域			
监管	能力	触点	运维
流程与政策 合规性 培训 供应链管理	攻击模型 安全设计方案与安全开发组件 第三方组件库管理 标准与要求 敏感数据处理	安全需求分析 安全设计 安全实现 安全测试	渗透测试 软件环境 运营支持 应急响应

图 1 S-SDLC CMM整体结构图

2.4. S-SDLC CMM评估域概述

S-SDLC CMM的各评估域、子域、相关安全实践以及评价指标框架如下。

监管域	
子域	实践
流程与政策	建立统一的企业软件安全战略
	制定组织安全开发管理流程
	建设和运营组织安全文化

	建立组织级 SDL 量化管理体系
合规性	合规性法律文件和要求转化为安全需求
培训	对高管进行软件安全意识培训
	对技术人员进行安全技能培训
	根据公司经验教训建立并使用特定的培训材料
	外部优秀安全经验吸收能力
供应链安全管理	成立软件供应链安全风险管理委员会
	建立正式的软件供应链安全管理制度

能力域

子域	实践
攻击模型	识别可能存在的攻击者
	建立攻击模型和技术知识库
	建设攻击团队
安全设计方案与安全开发组件	建立相对完善的安全设计库
	建立相对完善的威胁库
	安全开发组件
第三方组件库管理	建立并使用内部完整定义的第三方组件库
标准与要求	定义渗透测试标准
	建立一个有关安全信息的内网站点
	定义安全需求基线
	定义安全编码规范
	定义代码安全审核的标准
	定义第三方开源组件分析活动的标准
	定义安全红线
	定义安全测试标准
	定义数据库、中间件和操作系统的安全配置基线
	定义渗透测试标准

	定义漏洞管理标准
敏感数据处理	识别并标记敏感数据
	对敏感数据进行安全处理

触点域

子域	实践
安全需求分析	安全需求分析
安全设计	使用安全设计原则
	安全团队参与系统架构活动
	开展威胁分析
	根据安全需求进行安全设计
安全实现	执行代码安全审核
	执行安全编码规范检查
	对软件系统执行第三方组件安全扫描
安全测试	执行安全测试
	使用自研工具进行安全测试

运维域

子域	实践
渗透测试	执行渗透测试
软件环境	建立环境隔离策略
	建立统一的标准软件库
	实施系统安全基线加固方案
	实施安全的容器化部署方案
运营支持	建立拥有安全能力的运营支持小组
	维护操作环境规范说明
	建立持续监控机制
	持续优化安全策略

	建立安全补丁与更新管理流程
	建立持续审计机制
应急响应	建立拥有安全能力的应急小组
	制定应急响应预案

2.5. S-SDLC CMM成熟度级别

子域能力成熟度：S-SDLC CMM定义了5个成熟度级别以表示S-SDLC评估子域和组织能力成熟度，具体如下表：

表格 2 S-SDLC CMM 子域和组织能力成熟度说明

成熟度级别	详细说明	特征
1-基本执行	在这一级别，S-SDLC 子域内少部分的安全实践通常会被执行。但实践的执行可能未经严格的计划和跟踪，而是基于个人的知识和努力。同时，该安全能力并未完全内化到组织内。	随机、被动的安全开发过程
2-计划跟踪	在这一级别，S-SDLC 子域内的安全实践的执行是经计划并被跟踪的。同时，组织通过对安全实践进行测量来跟踪执行情况并保持对安全活动的管理。	主动、非正式的安全开发过程
3-充分定义	在这一级别，S-SDLC 子域内的安全实践按照充分定义的过程执行。充分定义是指软件安全开发流程实践有标准化、文档化的管理流程或经过裁剪并得到批准的一部分。这一过程与第 2 级（计划跟踪级）的主要区别在于安全实践的管理过程是组织级标准化、文档化的。	正式的、规范的安全开发过程
4-量化管理	在这一级别，S-SDLC 子域内的安全实践经过组织级地收集、分析和执行详细的测量活动，使组织获得对软件安全开发流程的量化理解和预测后续执行情况的能力。这个级别执行的管理是客观的，安全管理的质量是量化的。这一级与第 3 级（充分定义级）的主要区别在于对安全实践管理的定量理解和控制。	可量化控制的安全开发过程
5-持续优化和行业输出	在这一级别，S-SDLC 子域内的安全实践会基于组织长久的业务目标建立一个针对过程有效性和执行效率的量化目标，并围绕这一目标进行持续改进。这一级与第 4 级（量化管理级）的主要区别在于组织存在持续的优化过程。同时，由于安全管理过程是持续优化的，组织会实时向行业输出安全实践和安全能力，形成组织对行业的辐射和对外的影响力。	安全过程可持续优化，建立业界标杆

S-SDLC CMM

Part 02

安全开发实践

3. 安全开发实践

3.1. 监管

监管域描述了软件开发组织制定软件安全开发流程、合规要求、管理制度等的体系化文件。

3.1.1. 流程与政策

S-SDLC的实施必须是自上而下的，由高层授权，中层平衡相关利益方的诉求，从而达成组织对软件开发安全的共识，为后续的安全实践落地提供合适的土壤。组织需要对软件安全开发的流程和合规性进行监管，保证执行的效果，最终将安全内化到企业文化之中，为企业的业务产品提升安全方面的核心竞争力。“流程与政策”包括建立统一的软件安全战略、制定组织安全开发管理流程、建设和运营组织安全文化、建立组织级别的SDL量化管理体系等安全实践。

3.1.1.1. 建立统一的软件安全战略

从软件开发安全的角度来看，统一的软件安全战略是组织完善安全开发流程而确定的发展方向以及长期计划。

除此之外，企业拆解安全战略为实施规划时，组织应确定这些实施规划与组织的安全目标保持一致。

建立软件安全战略方法举例：

- 对业务系统按照法律法规要求进行分类分级。
- 建立不同等级和分类系统的安全目标。

3.1.1.2. 制定组织安全开发管理流程

重视软件安全的组织，往往会制定一套指导安全软件开发的流程，使得软件安全活动有序、高效地开展。为此，他们根据组织自身的需要定义安全活动，植入到现有的如瀑布、敏捷等研发流程中，打造成为符合企业自身文化的软件安全开发流程。该流程不仅要包含清晰的角色职责、活动描述、输入输出和对应模板，还应该在组织内部公布该流程和相关计划，使其得到干系部门的支持。另外，随着组织的发展，流程在运行过程中还需要不断更新。

3.1.1.3. 建设和运营组织安全文化

软件开发组织应该设定负责安全文化推广的宣传角色，并通过在组织内部宣传和推广安全软件开发流程，使研发各部门了解软件开发组织的软件安全战略及相关的安全目标和流程，让安全团队和开发团队形成更好的互动关系。除此之外，企业还应具备文化输出的能力，将软件开发安全的实践和文化对外输出，提升行业影响力。

3.1.1.4. 建立组织级别SDL量化管理体系

软件开发组织的SDL流程运转在达到一定成熟度的时候，应该建立统一的S-SDLC管理系统量化安全活动的执行，在提升S-SDLC流程效率的同时对其进行改进和优化。为此，需要建立可度量的SDL过程质量目标以及S-SDLC结果质量目标，然后建立统一的SDL管理平台统计、计算和展现这些质量目标；其次，需要针对质量过程及结果不达标的情况进行根因分析并制定相关的改进措施；最后，应该确定S-SDLC的度量目标，有必要则对其进行更新。

3.1.2. 合规性

研发初期就应该从需求层面考虑软件应该符合哪些相关的法律法规，对于合规性的漠视可能会让组织付出惨痛的代价。“合规性”主要描述了软件开发组织从需求就开始重视监管规则。 该子域包含了转化政策法规、行业条例、监管要求为安全需求的安全实践。

3.1.2.1. 合规性法律文件转化为安全需求

软件开发组织的软件产品都面临行业法律、法规以及监管政策的约束，为了使软件开发组织的软件产品合法、合规，应该安排专人或部门，负责解读国家、行业的法律合规文件，并将其转化为安全需求，然后纳入软件产品的开发计划中。

3.1.3. 培训

安全知识的培训对软件开发组织中的每个角色都至关重要，这是做好软件开发安全实践的前提。“培训”域主要描述了组织为提升高管和技术人员安全知识水平而制定并更新软件安全培训内容和考核内容的能力。

3.1.3.1. 对高管进行软件安全意识培训

高管一般指的是组织中C(Chief)字头的管理者，拥有软件开发组织制度建设和资源调配的最大权限。高管在软件安全上的认知对推进组织的软件安全建设将起到决定性的作用。通常情况下，高管一般是非安全人员或技术人员，对他们应该着重强调意识培训及案例、法规的解读，帮助他们认识到软件安全的重要性。在该子域中我们将评估高管入职培训或者是定期培训的频率以及相应的考核内容，确保高管能建立起足够的软件安全意识。

3.1.3.2. 对技术人员进行安全技能培训

技术人员是软件开发的实施者，对软件安全有着最直接的影响。本着“安全无处不在，安全与业务融为一体”的理念，软件安全的知识技能应该覆盖到整个软件开发生命周期中。涉及到软件开发生命周期的相关人员，都应该进行对应的安全知识技能培训，培训的内容应该在高管意识培训的基础上，增加具体的安全技能知识，包括常见的安全攻击方式、安全代码规范等，根据产品经理、项目经理、开发、测试等不同角色制定不同内容，定期培训和考核，并将考核结果与绩效指标挂钩，让技术人员提升软件安全的认知。

成熟的软件开发组织会建立各研发角色与安全能力等级的对应关系，比如开发工程师要求具备初级安全能力，架构师要求高级安全能力。这些能力的建设需要通过长期大量的培训积累而来。

3.1.3.3. 根据公司经验教训建立并使用特定的培训材料

软件开发组织常常会将组织或项目管理运作过程中吸取的经验教训用作内部培训的材料。但由于每个项目的独特性，很难从某个技术方案中提炼出能被其他项目组采纳的内容。而软件安全问题具有普遍性，它是真实发生过的案例，并且也可能发生在其他组织中，用作培训材料更容易对其他组织起到警醒作用。在安全测试、渗透测试或攻防演练等活动中发生的安全事件，都是软件开发组织重要的知识和经验，通过对其不断总结和复盘，形成培训材料，将有助于提升相关角色的安全意识和技能。

3.1.3.4. 外部优秀安全经验吸收能力

外部优秀的安全经验能够让组织感受到所在行业软件安全开发能力的发展趋势，以及组织在行业中所处的位置，使得组织的管理者能更客观地观察到自身软件存在的安全问题。定期聘请外部经验丰富的专家为组织提供咨询或培训服务，有助于组织了解新的安全思想、安全技术以及安全实践，紧跟软件安全发展的最新趋势，提升团队的安全开发能力。外部的优秀经验可以来自于有影响力的个人或者组织，通过宣讲会、驻场、咨询、会议等提升行业软件安全能力的指导性影响。

3.1.4. 软件供应链安全管理

在数字化转型的大背景下，软件开发组织采购了大量的第三方软件加快组织的数字化转型，随之而来的是软件供应链安全攻击事件地成倍增长，造成了越来越严重的危害。供应商软件产品的安全性对软件开发组织而言至关重要。“软件供应链安全管理”主要是帮助企业建立正式的制度，管理软件供应链的安全风险，该子域的内容包括成立软件供应链安全风险管理委员会和建立正式的软件供应链安全管理制度。

Argon Security 公司的一项研究显示，相比2020年，2021年发生的软件供应链攻击事件增长了300%以上。例如“金链熊事件”致使200家重要机构受害。

3.1.4.1. 成立软件供应链安全风险管理委员会

传统的供应链安全风险由采购团队执行管理，信息安全团队仅提供网络安全相关的要求，由于参与的角色相对较少，安全风险无法有效地管控。软件开发组织应成立供应链安全风险管理委员会，委员会的成员应来自不同的部门，确保供应链安全风险能够得到及时、有效的反馈和处理。

3.1.4.2. 建立正式的软件供应链安全管理制度

软件开发组织与软件供应商合作时，应该保证软件供应商符合组织在安全和控制措施方面的要求。同时，在签订合同时应明确各自的权责、处罚和追责措施等。软件开发组织内部也应针对供应商制定相应的风险审查和缓解流程，确保在复杂的业务环境下影响组织业务的连续性内部可控。

3.2. 能力

能力域描述了软件开发组织保障各种安全活动执行所具备的基础设施。

3.2.1. 攻击模型

软件开发组织应能主动了解可能存在的攻击者、攻击模式和相关技术，做到知己知彼、防患未然。“攻击模型”主要描述了当前软件开发组织需要具备的建立攻击者视角的主动防御能力。

3.2.1.1. 识别可能存在的攻击者

大部分组织未对潜在的攻击者进行有效的识别，处于被动防御的状态，只有在被攻击之后才会对攻击者进行分析，识别安全弱点，这样可能会增加安全响应的成本和组织的安全风险，最终对业务的连续性造成影响。软件开发组织应该从不同的角度识别潜在的攻击者，通过创建潜在攻击者清单帮助组织识别面临的攻击面，为安全设

计提供更全面的分析场景。

3.2.1.2. 建立攻击模型和技术知识库

软件开发组织可以通过内部渠道或外部渠道收集各种攻击情报和攻击案例，通过分析攻击情报和攻击案例及时发现并处置攻击造成的危害，帮助组织及时止损。收集到攻击情报和攻击案例之后软件开发组织可从攻击方式和攻击影响等方面多维度地对其进行剖析，建立相应的渠道例如论坛、社区、邮件等进行讨论。这些互动方式有助于开发人员、测试人员了解攻击情报、攻击原理、漏洞细节，从而提升他们的安全能力和意识，并为安全设计提供支撑。

3.2.1.3. 建设攻击团队

软件开发组织可建设一支自主可控的攻击队，不断研究新的攻击模式和攻击方法，以此来提升组织的攻击防范能力，从被动转为主动，保持组织安全建设的先进性。同时也可不断地对组织进行攻击演练，检验组织的攻击防范能力和应急响应处理能力，发现组织的相关弱点，并针对弱点进行改进。

3.2.2. 安全设计方案与安全开发组件

软件开发组织可以通过安全设计方案和安全开发组件减少研发团队对安全专业知识的依赖，快速实现对研发团队的安全赋能。“安全设计方案与安全开发组件”描述了软件开发组织构建安全开发、安全设计、威胁分析知识库和安全开发组件的能力，“安全设计方案与安全开发组件”的使用可以提升安全开发效率，降低安全开发成本。

3.2.2.1. 建立相对完善的安全设计库

软件开发组织应该针对软件系统常见且重要的安全需求建立安全设计库，降低开发人员安全设计的难度，例如：给认证、防注入、文件操作和安全传输等重要的安全需求提供安全设计建议；设计不可猜测的用户ID；调用后台接口需要用户凭据（Token）；避免拼接SQL语句；校验文件类型/大小/文件名等。

3.2.2.2. 建立相对完善的威胁库

威胁分析可以从攻击者的视角发现软件系统的潜在风险。基于软件系统的类别和系统架构建立威胁库将有助于威胁分析活动。威胁库通常需要配合威胁分析工具去使用，如何最大限度的实现自动化生成威胁是威胁分析工具及威胁库需要考量的问题。

3.2.2.3. 安全开发组件

软件开发组织应该建立专门的团队，构建并发布安全开发组件供产品研发团队使用，提升安全开发效率。安全开发组件示例有：身份验证、角色管理、密钥管理、日志记录、密码、协议、防SQL注入解决方案、防CSRF解决方案、会话保持机制等。

3.2.3. 第三方组件库管理

软件系统的开发往往会用到大量的开源组件，软件开发组织需要完善流程、加强安全管控，减小第三方组件库的潜在安全风险。“第三方组件库管理”描述了软件开发组织管控开源组件的能力。

3.2.3.1. 建立并使用内部完整定义的第三方组件库

开发软件不可避免地要使用大量第三方组件，第三方组件的选择、下载、安装和更新等各环节对软件系统的安全性都可能产生重要影响。软件开发组织需要安排专门的治理团队按照正式的流程建立、更新和维护内部的第三方组件库。组件库应能满足所有产品线的开发需求，如果组件缺失，可以通过正式流程申请、批准和新增组件。软件开发组织应该管控各个开发团队的组件使用，使得开发只能从内部组件库下载和使用第三方组件，禁止通过互联网下载或者私自拷贝等方式引入第三方组件。软件开发组织应该定期对第三方组件库进行安全扫描，对于发现的安全隐患能及时整改。

3.2.4. 标准与要求

“标准与要求”描述了安全活动的标准以及要求是如何定义的。在这个子域中我们对于安全开发流程中可能涉及到的规范标准都做了定义描述。相关实践包括：定义安全需求基线、建立安全编码规范以及定义代码安全审核标准等相关标准和要求。

3.2.4.1. 建立一个有关安全信息的内网站点

企业应该建立安全门户网站供相关角色获取组织实时发布的安全政策及安全资料，内容可以包括：SDL及相关的流程规范、软件安全培训课程、软件安全最新趋势等。企业应该安排人员在组织内部推广该网站，使相关研发角色了解该网站并从其中获取安全信息，除此之外，还应定期评估并更新该网站的内容。

3.2.4.2. 定义安全需求基线

安全需求基线指的是软件开发组织内所有软件系统开发项目都需要遵循的安全需求和要求。不同的软件系统安全需求基线应该有所差别。软件开发组织制定的安全需求基线应该包含：通用安全需求、软件系统特定安全需求、法律法规以及行业监管等内容。针对软件系统特有的安全需求基线，应该考虑到开发语言和其使用的技术框架。

3.2.4.3. 定义安全编码规范

软件开发组织应该根据软件系统使用的开发语言和开发框架，制定相应的安全编码规范。安全编码规范包括但不限于：规避常见漏洞的安全编码习惯（例如：防SQL注入、防XSS），开发语言相关的安全编码规则，开发框架相关的安全编码规则（例如：Spring、Mybatis）。

3.2.4.4. 定义代码安全审核的标准

代码安全审核指的是在软件项目开发过程的每次代码转测前对当次增量代码进行的安全审核，确保新增代码遵循安全编码规范，同时发现新增代码的安全漏洞；代码安全审核可采用人工检查或人工与自动化工具相结合的方式执行。软件开发组织应该制定代码安全审核的标准，主要包含对安全编码规范的审核内容，以及审核流程、角色及职责等。

3.2.4.5. 定义第三方开源组件分析活动的标准

第三方组件安全是保障软件系统安全的重要内容，软件开发组织应该定义第三方开源组件分析活动的标准，

内容应该包括定义不同漏洞的风险等级、定义各种开源组件许可协议的相关法律风险等级，分析活动的角色、职责和通过标准等。

3.2.4.6. 定义安全红线

安全红线指的是软件系统必须在上线前修复的漏洞类型列表。软件开发组织应该为每一类软件系统定义不同的安全红线，且在研发流程中规定，软件必须在安全红线中所有的漏洞修复完成后才能进行版本发布。

3.2.4.7. 定义安全测试标准

安全测试是保障软件系统安全的重要手段，软件开发组织应该定义安全测试的标准，内容包含但不限于：要求测试人员针对安全需求编写安全测试用例并确保安全用例覆盖安全需求；要求对测试发现的漏洞进行全生命周期的跟踪管理；确定安全测试活动的角色、职责和通过标准等。

3.2.4.8. 定义数据库、中间件和操作系统的安全配置基线

为确保软件环境的安全，软件开发组织应该为用到的数据库、中间件和操作系统定义安全配置基线，该基线在生产环境中验证有效后应该作为标准长期使用。软件开发组织还需要定期评估安全配置基线内容，并对其进行更新。

3.2.4.9. 定义渗透测试标准

软件开发组织应根据自身的实际情况定义各软件系统的渗透测试需求、情报收集方式、漏洞利用方式等，包括：需要渗透测试的软件系统及版本类型、渗透测试的频率、渗透测试的工具、活动和具体要求。

3.2.4.10. 定义漏洞管理标准

关于具体漏洞类型的定级，目前行业还没有明确统一，业内通常根据漏洞的利用概率、利用后的影响以及法律风险等因素来定义漏洞的分级标准（如CVSS标准）。软件开发组织应该根据业务和软件系统的具体情况，参考行业标准，定义适用于自身的漏洞分级标准以及相应的处理办法。漏洞的范围应涵盖：安全测试发现的漏洞、软件成分分析发现的漏洞以及渗透测试发现的漏洞等。良好的漏洞管理实践有：建立统一漏洞管理平台，对漏洞进行全生命周期管理，追溯已发现的漏洞并进行根因分析，然后对相关流程进行改进。

3.2.5. 敏感数据处理

为了保护敏感数据的安全性，软件开发组织需要采取一系列措施，如对敏感数据进行加密、安全传输、安全存储、密钥管理、数据脱敏等，确保敏感数据不被非法获取。加密是保证敏感数据安全性的基本手段，即使加密过的敏感数据被非法泄露到外部，也能保证内容不可读。在此基础上，将加密过后的敏感数据存储在安全的位置，并对密钥进行管理也是保护敏感数据机密性的核心工作，这将直接影响到敏感数据的安全性，因此敏感数据的访问控制和备份以及密钥的生成、存储、分发、更新和销毁，组织都必须予以重视。在数据传输方面，通常采用HTTPS或TLS等安全协议，以确保数据在传输过程中不被窃取或篡改。数据脱敏可以在保证数据可用性的前提下，隐藏或部分显示敏感数据，以保护个人隐私和数据安全。鉴于敏感数据安全处理需要从多个方面进行考虑，组织应该综合实现敏感数据从生成、传输、存储到使用的完整安全处理流程。

3.2.5.1. 识别并标记敏感数据

软件系统中通常包含某些敏感数据，例如：身份证、手机号、银行卡以及微信号等，这些敏感数据都应该得到保护。软件开发组织应该结合自身业务和产品，定义敏感数据清单，然后构建针对敏感数据的识别和标记功能，比如将“phone”、“pwd”、“weixin”、“cardno”等字段标记为敏感字段，使得软件系统在遇到敏感数据的时候，能对其进行加密、脱敏等安全处理。

3.2.5.2. 对敏感数据进行安全处理

软件系统在收集用户输入信息、调用其他程序、处理和存储数据的过程中，都可能因为安全功能缺失或软件系统本身的漏洞而导致敏感数据泄露。因此，软件开发组织可适当增加加密保护措施，保证传输信道和传输数据的安全。应建立对传输数据完整性的检测机制，并具备数据容错或恢复的技术手段，应部署能审核及监控信道安全配置、密码算法配置、密钥管理等保护措施的技术工具。例如：建设KMS密钥管理系统，提供密钥管理和解密接口，以用于敏感数据的安全传输和存储。

3.3. 触点

软件开发生命周期当中的每个阶段都有对应要处理的安全活动，如果我们把它们单独拿出来看，似乎增加了很多额外工作，但其实大多数活动本应该视为是原有开发活动的组成部分。触点域描述了在软件开发生命周期中应该如何融入安全实践。

3.3.1. 安全需求分析

安全基线规定了软件系统在安全、合规以及漏洞方面的基础要求，软件开发组织需要按照组织定义的安全基线，识别每一项业务需求，并将其转化为安全需求。安全需求的例子包括：为新增的重要业务功能采用多因素的方式来认证身份，收集敏感数据时弹窗提示用户收集的数据及其用途，为WEB应用订单的提交增加预防WEB安全漏洞的功能等。建立并维护一套安全需求知识库，将有助于团队准确而高效的识别安全需求。 安全需求知识库的建立是一个长期的过程，并且要根据新的业务场景、使用新的技术手段，保持安全需求知识库的时效性。

3.3.2. 安全设计

“安全设计”包括安全需求所引导的安全设计，以及从攻击者角度出发的安全威胁分析，还有软件层次的整体性架构设计。

3.3.2.1. 使用安全设计原则

软件系统的架构设计要考虑性能、高可用和扩展性，还有安全性。软件系统架构设计应遵循一些基本的安全设计原则，如攻击面最小化、基本隐私、权限最小化、纵深防御等，提升软件系统的防御能力。

3.3.2.2. 安全团队参与系统架构活动

安全团队应参与系统架构活动，包括所需解决问题的分解、模块的划分、组件的选用以及组件协作等内容，也应参与架构的具体方案设计如身份认证、授权及访问控制、密钥管理、敏感信息存储、加密传输、输入校验和日志记录等。

3.3.2.3. 开展威胁分析

威胁分析是指从攻击者的视角来识别软件系统可能面临的威胁，例如：身份认证系统是否有可能被绕过，管理员默认密码未修改、授权和访问控制系统是否存在缺陷导致越权以及是否存在独立可审计的操作日志等，然后对已识别的威胁设计缓解方案。通过威胁分析，组织可以识别潜在的威胁并采取相应的措施来强化安全性，同时还可以提高安全意识和培训开发团队，以更好地保护组织的信息和资产。

3.3.2.4. 根据安全需求进行安全设计

安全需求由业务需求面临的安全、合规等要求转化而来，与常规需求一样，安全需求需要对应的安全设计，比如为重要的操作（转账、支付等）设计不可猜测的用户ID，为用户输入设计相应的校验规则，或者引入安全开发组件等。为了开发团队能高效地进行安全设计活动，组织还可以建立和维护一套安全设计知识库。安全设计完成后，可以邀请资深开发工程师、架构师或者安全工程师对设计方案进行评审，确保安全设计满足已识别的安全需求。

3.3.3. 安全实现

安全实现包含了安全代码审计和安全编码两部分，安全编码指在安全设计的指引下进行代码的实现，安全代码审计是为了防止非预期的漏洞被遗漏。

3.3.3.1. 执行代码安全审核

采用自动化的静态代码扫描工具，能够高效率地对代码进行一些常见的安全漏洞扫描，比如XSS、SQL、XXE等。敏捷开发环境下，软件开发组织应该将自动化静态代码扫描工具与代码管理平台（GitLab、SVN等）或DevOps平台集成，实现提交代码合并请求时自动触发安全检查，如发现高危漏洞则拒绝合并，进而帮助开发人员快速修复安全漏洞。

相比纯自动化扫描，将自动化与人工相结合更能保障代码地质量。软件开发组织对代码进行人工检查，可以发现自动化工具难以识别的漏洞，例如越权操作、文件上传下载等逻辑类漏洞。除此之外，结合审查人员对系统业务的了解，对于一些关键功能的代码也可人工进行检查，例如加解密方式、敏感数据操作等，进一步保障编码的安全性。

3.3.3.2. 执行安全编码规范检查

安全编码规范对于很多软件开发组织仅仅是一份文档，即使做过一两次宣导，起到的作用也很有限。安全编码应该在安全需求、安全设计的指引下进行，如此，安全编码规范就自然而然地成为开发人员需要参考的文档。

软件开发组织应对安全编码规范的落地情况进行审查，可以采用以自动化静态扫描工具为主、人工为辅的方法。自动化工具可以将它嵌入到研发流水线中，并实现自动阻断。对于独特业务或者技术框架相关的编码规范，软件开发组织需要通过自定义规则或对工具进行二次开发等方式去实现覆盖。

3.3.3.3. 对软件系统执行第三方组件安全扫描

软件开发需要用到大量的第三方组件，组件的安全也应引起重视。对于第三方组件的安全，需要重点关注组件本身已发现的安全漏洞和已使用的许可协议。但由于第三方组件的数量越来越多，为快速定位到有风险的组

件，软件开发组织可采用自动化的方式对组件进行扫描分析，将发现的问题录入漏洞管理平台进行跟踪管理。

3.3.4. 安全测试

“安全测试”是基于安全需求引导而来的对于安全实现的确认，其中包括工具的使用以及测试结果的评估。

3.3.4.1. 执行安全测试

安全测试是基于软件需求说明书中关于安全功能需求的测试，测试的内容主要是：软件的安全功能实现是否与安全需求描述一致。软件开发组织应安排测试人员编写安全测试用例、执行测试以及跟踪管理bug和漏洞，并建立和维护一套安全测试用例库，提升测试人员的测试效率。

3.3.4.2. 使用自研工具进行安全测试

在测试阶段，使用IAST/DAST检测工具对软件系统的缺陷进行查找，可大大提升测试效率。但对于安全能力成熟度较高的软件开发组织，通用的解决方案已无法满足其复杂的安全需求，需要针对组织自身的技术架构，结合实际业务自研出合适的安全测试工具。此项活动的安全能力要求较高，需要依赖组织整体的安全能力建设。

3.4. 运维

运维域描述了软件开发组织的生产环境和软件上线需要执行的安全实践。

3.4.1. 渗透测试

“渗透测试”是上线前对软件残余风险的最后确认。

3.4.1.1. 执行渗透测试

渗透测试是指从攻击者的角度，使用工具加人工测试的方法，尽可能发现软件系统本身的漏洞和操作系统、中间件、网络环境的安全配置漏洞。通过利用这些漏洞，还原软件系统面临的安全风险，评估漏洞产生的影响，然后输出渗透测试报告。执行渗透测试前需要明确渗透测试的流程、人员和方法，测试结果通过漏洞管理平台反馈给工程团队，由他们在平台上发布流程并做出响应。对于需要定期执行渗透测试的上线后的系统，根据其安全等级确定上线后的测试频率。

3.4.2. 软件环境

“软件环境”描述了软件在运行环境需要考虑的安全基线并根据基线的内容对环境进行检测和加固。

3.4.2.1. 建立环境隔离策略

随着网络攻击手段地持续升级，组织需要建立相应的隔离策略，保证信息和资源在可信网络和非可信网络之间实现安全交换和共享，保证信息交换和资源共享的机密性、完整性、可用性。例如：建立公网和私网的隔离，通过网关控制来减少应用的攻击面。同时也需要随着业务的不断开展，以及组织的变动及时地调整相应的隔离策略，保证业务的高可用性。

3.4.2.2. 建立统一的标准软件库

软件开发过程中需要第三方软件或系统作为组织软件业务系统的载体。为避免直接从公网下载未经验证的第三方软件或系统引入不可控的安全风险和法律风险，软件开发组织应建立内部的标准软件库、系统库并进行统一管理。根据业务情况，各产品线可从标准库中选择合适的操作系统和软件，满足自身高效部署、高可扩展和安全的要求。随着操作系统和软件的不断更新，组织应定期对系统库和软件库进行安全性评估和版本归并。

3.4.2.3. 实施系统安全基线加固方案

数据库、中间件和操作系统默认安装时存在部分不安全的配置，例如默认账户、弱口令、不安全的权限配置等。为保障软件开发组织使用的数据库、中间件和操作系统满足“默认安全”的设计理念，应制定对应的安全基线加固方案。方案的制定可围绕云安全配置、操作系统安全配置、数据库安全配置等方面展开，例如关闭无用端口、配置口令、最小化权限等。同时也应定期对数据库、中间件和操作系统进行安全加固检查，评估实施结果。

3.4.2.4. 实施安全的容器化部署方案

为满足快速迭代的需求，越来越多的软件开发组织会将应用程序拆分成多个微服务进行开发。但传统的运维部署方式无法满足微服务下的部署要求，于是，具有高效、高可扩展、敏捷优势的容器镜像和编排技术应运而生。但容器化技术也并非只有好处，它仍有相应的安全风险，需要通过实施安全的容器化部署方案来保障应用安全，例如创建受信任的基础镜像、漏洞扫描、容器架构安全配置等。

3.4.3. 运营支持

“运营支持”描述了软件开发组织以业务安全为目标，保障应用安全运营的相关能力。相关实践包括建立运营支持小组、维护操作环境规范说明、建立持续监控机制和持续优化安全策略等。

3.4.3.1. 建立拥有安全能力的运营支持小组

为了确保软件部署上线后出现的安全事件能有效处置，安全措施有效到位，软件开发组织应具备一支拥有安全能力的运营支持小组支撑软件开发后续的运营工作。与普通运营人员不同，该小组还需要负责安全运营的工作，尤其需要识别和定义软件可能面临的风险，明确风险发生时如何正确处理，有效实施安全运营工作，合理增强各业务应用系统的稳定性、可靠性和高效性，提升用户满意度。

3.4.3.2. 维护操作环境规范说明

软件开发组织应制定并维护操作环境规范说明，保证运营管理工作具有相应的操作指导。操作环境规范说明主要用于规范运营人员对操作环境进行变更时的行为，避免因运营人员的错误配置等产生新的安全弱点，如默认账户、口令、不安全的权限配置等。

3.4.3.3. 建立持续监控机制

软件开发组织应当在软件运行期间持续监控，通过监控信息来判定软件是否存在异常，包括CPU内存占用率高、系统死机等情况。除了使用安全设备实现监控及异常行为的告警外，软件开发组织还可以收集外部渠道的威胁情报数据，建立网上安全应急响应中心，接收白帽子提交的安全漏洞信息，更精确地发现系统面临的威胁。

3.4.3.4. 持续优化安全策略

在如今网络业务复杂，攻击者入侵手段变幻莫测的情况下，安全策略出现冗余、异常甚至失效的可能性很高，为了保证已部署的防火墙、IDS、IPS等能与时俱进，具备更加全面的防护能力，软件开发组织应结合实际情况持续更新和优化相应的规则和措施。

3.4.3.5. 建立安全补丁与更新管理流程

要想更全面地保证软件运行时的安全，不仅要关注软件本身的安全，还要关注软件运行环境的安全。软件开发组织应建立应用程序安全补丁与更新管理的活动流程，明确各管理活动的触发条件等。在此基础上，组织还应持续关注软件本身及其用到的第三方组件在CNVD、CVE等渠道发布的安全预警，将发现的漏洞输入到漏洞管理平台中，通过管理流程跟踪，及时主动地修复已暴露的漏洞，将安全风险降到最低。

3.4.3.6. 建立持续审计机制

软件开发组织应规范系统日志的管理，定期对日志进行审计。日志一般会记录系统状态、用户IP、操作业务、访问日期等信息，对日志进行审计，有助于发现程序自身的安全缺陷，并为安全事件发生后的成因分析提供线索。审计机制建立后，还可以对某些关键的系统或业务定期进行基于审计日志的大数据分析与事件挖掘活动，主动发现尚未告警的安全事件或隐患。

3.4.4. 应急响应

“应急响应”描述了软件开发组织对安全事件的响应、处理和恢复能力。其实践包括建立应急小组和制定应急响应预案。

3.4.4.1. 建立拥有安全能力的应急小组

软件开发组织应建立安全响应团队，在信息安全事件发生时依照明确的应急响应预案对信息安全事件进行处置。该团队负责定义、收集、评估事件响应过程中的指标，并根据指标对安全事件做出相应处理。为保证响应的及时性，团队应建立与内部或外部软件开发组织沟通的渠道，保持适当的联络和信息共享，保证安全事件发生时的跨部门合作，快速定位和解决问题。

3.4.4.2. 制定应急响应预案

软件开发组织应依照明确的应急响应预案来防范及应对信息安全事件，避免采取临时的应急措施而导致事件影响的进一步扩大。预案包括对事件进行分级分类，制定相应的应对计划等；除此之外，应急响应团队还应定期组织安全事件应急演练，校验预案的有效性，演练结果可作为预案不断改进的依据。在演练和实际响应的过程中，安全响应团队应对流程的各个环节做好文档记录，包括处理者和其负责的事项，事件的成因分析等，用于事件的回溯。

4. S-SDLC CMM 线上社区

http://s-sdlccmm.com

S-SDLC CMM的线上社区能让您更加近距离的感受模型的使用，通过线上测试、与成员交流互动等方式，可以让我们更加清晰了解如何高效使用该模型，以帮助企业提升软件开发过程的能力成熟水平。

5. 特别鸣谢

特别感谢深圳开源互联网安全技术有限公司对该项目提供的支持。如有疑问，请联系：ricky.xu@owasp.org。