# SDL/DevSecOps

## 不足

两种理念都是针对系统/应用/产品的安全理念，是围绕着应用的生命周期各阶段展开的。并不是针对公司的安全体系建设理念。

安全理念需要落地的话，有的方面需要安全制度及流程的配合才能落地，而传统解读中，关于安全制度及流程的建设比较少。

各行业各公司对于信息安全，网络安全的核心关注点也有不同，如互联网行业，传统行业，高端科技行业，他们的核心资产跟关注点就不一样。

因此安全体系建设，不能按照死板的安全理念进行落地，要根据公司实际情况，进行因地制宜的整理规划。优先发现行业，公司面临的安全风险，梳理公司关注的核心资产，构建出人-资产-流程系统安全管理及技术体系。从各种理念中吸收，融合，发展出适合本公司的安全建设理念，划分出合适的优先级。

当方案决策之后，针对某产品/应用具体实施安全建设的时候，此时才适合应用SDL/DevSecOps 理念进行安全建设。

## 差异



SDL，安全开发生命周期， 是微软提出并 于 2004 年正式应用的。

然后2006年，微软发布了基于SDL流程的第一个操作系统 Windows vista。

SDL 是微软基于其本身情况而产生的安全理念，符合自身，但是不一定符合其他公司。因为微软的产品就是操作系统，其具有长周期性、一次性产品、变更少等特点，所以产生了SDL的理念，不是DevSecOps理念。微软的产品特性导致其对安全前置的要求非常高，因为产品不具备实时修复实时更新能力。

SDL理念是由安全部门主导产品的生命周期，而不是由产品经理主导。安全在产品之初就在把控各节点的产品安全性，并且具有主导权，决定权。

只有这样才能真正落实SDL理念。安全的主导权是SDL的先天条件。

微软的安全部门获得这个条件的原因是，XP系统及之前系统的巨多的漏洞，对微软造成了极大极大的影响。

安全部门的主导权，跟业务产品的特点，决定了SDL在微软的形成发展，也决定了其他公司不一定有条件复刻微软的SDL体系。

所以好多其他公司在用SDL理念进行安全介入的时候，就产生水土不服的感觉，感觉SDL不好落地，不容易施行。

后来，产品，技术本身等根本性的因素也发生了很多变化。

产品形态变化：传统系统，单机软件，到web系统到APP。

开发理念变化：瀑布模型，到敏捷模型，到devops模型。

开发技术栈变化：从混沌，到MVC，到微服务，到云原生。其技术栈本身也添加了很多安全理念。

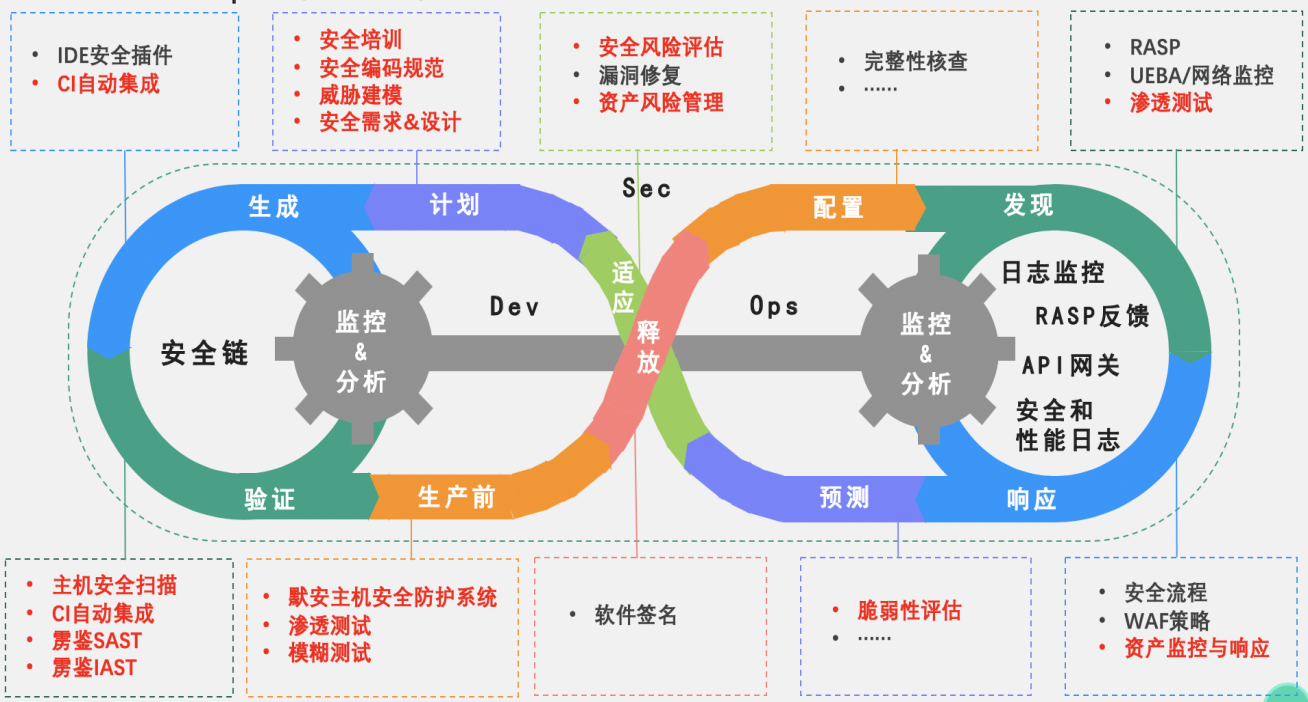
运维技术栈变化：自动化，CICD的出现，云原生架构的出现。

这些变化导致，产品的安全修复成本，从最初的难以修复，变成了容易修复。修复难度在整个生命周期中是拉平的。

最初的开发技术栈导致，SDL是从安全培训开始，在前期，威胁建模在理念中占据了很大的篇章。因为产品架构在后期修改的话，成本极其巨大。所以要通过威胁建模之后，设计出来安全的产品架构，才能进行下一步的实施阶段。

但是现在的开发及运维技术栈，其本身就自带了高可用，可扩展，安全性等等特性。如MVC框架，如K8S，其理念是不断更新的。更改架构的成本也在慢慢变低。

所以后来有公司提出了 DevSecOps的应用安全理念。



DevSecOps理念中，只要求安全尽早参与，并没有要求安全的主导权。没有侵占产品经理的权力。Devsecops也完全覆盖了SDL的内容，并有所改进，加入了自动化的理念，将安全融入到开发测试运维的自动化的流程中去。

SDL理念中，没有持续修改，持续运营，持续监控的内容，因为操作系统是一次性产品，本身就没有这些要求。新版本按SDL走一遍，旧系统可以直接放弃维护。

DevSecOps理念中，就有持续的概念。对于无法重构的已上线的产品，安全工作可以有持续的安全监控，安全测试等手段来提高产品的安全性。

有的行业SDL看起来也适合，比如游戏行业。但实际上还是有巨大差别的，一个是进程的主导权，一个是持续运营运维。

游戏的开发周期很长，跟系统类似，但是游戏的运营周期更长，系统可以直接放弃维护，但是游戏不能放弃维护。所以DevSecOps理念也更适合游戏。

所以安全理念在实际公司中的应用，应该是跟产品现状紧密相关的。

应当根据公司产品，根据工作客观条件，来使用，甚至重新拼凑构建一个适合自己公司产品的应用安全实施内容，进而来落地应用安全体系建设。