Weighed Down by Legacy

Large financial organizations, like other enterprises, have typically

been built up over years through mergers and acquisitions. This has

left them managing huge application portfolios with thousands of

different applications, and millions and millions of lines of code, in

all kinds of technologies. Even after the Y2K scare showed enterprises

how important it was to keep track of their application portfolios,

many still aren’t sure how many applications they are running,

or where.

被遗留系统所拖累

像其他企业一样，大型金融机构通常是通过多年的兼并和收购建立起来的。他们管理着成几千个应用组成的庞大应用组合，这些应用由各种技术开发，有着数百万行的代码。即使千年虫恐慌已经显示企业跟踪他们的应用程序组合有多重要，许多企业仍然不确定他们正在运行多少应用程序，或者在哪里运行。

Legacy technology problems are endemic in financial services,

because financial organizations were some of the earliest adopters of

information technology. The Bank of America started using auto‐mated check processing technology back in the mid 1950s. Instinet’s

electronic trading network started up in 1969, and NASDAQ’s computerized

market was launched two years later. The SWIFT international

secure banking payment network, electronically linking banks

and payment processors for the first time, went live in 1977, the

same year as the Toronto Stock Exchange’s CATS trading system.

And the “Big Bang” in London, where the LSE’s trading floor was

closed and the UK financial market was computerized, happened in

1986.

遗留技术问题是金融服务领域普遍存在的问题，因为金融机构是信息技术的最早的一批拥趸。美国银行早在20世纪50年代中期就开始使用自动支票处理技术。电子交易网络始于1969年，纳斯达克的计算机市场则在两年后推出。SWIFT国际安全银行支付网络在1977年首次投入使用，电子地连接起银行和支付机构。同年，多伦多证券交易所的CATS交易系统也投入使用。接着是1986年伦敦的“大爆炸”，伦敦证交所交易大厅被关闭，英国金融市场实现计算机化。

Problems with financial systems also go back a long time. The

NYSE’s first big system failure was in 1967, when its automated

trade reporting system crashed, forcing traders to go back to paper.

And who can forget when a squirrel shut down NASDAQ in 1987?

金融系统的问题也可以追溯很久。这个纽约证交所的第一次大系统故障是在1967年，当时它的自动化系统交易报告系统崩溃，迫使交易者重新使用书面交易。谁能忘记1987年松鼠关闭纳斯达克？

There are still mainframes and Tandem NonStop computers running

business-critical COBOL and PL/1 and RPG batch processing

applications in many large financial institutions, especially in the

back office. These are mixed in with third-party ERP systems and

other COTS applications, monolithic J2EE systems written 15 years

ago when Java and EJBs replaced COBOL as the platform of choice

for enterprise business applications, and half-completed Service Oriented

Architecture (SOA) and ESB implementations. Many of these

applications are hosted together on large enterprise servers without

virtualization or other effective runtime isolation, making deployment

and operations much more complex and risky.

在许多大型金融机构中，仍有大型机和Tandem不间断计算机在运行这关键业务地COBOL和PL/1以及RPG批处理的应用，尤其是在后台。它们与第三方ERP系统、其他COTS应用，编写15年以前单体的J2EE系统，以及半成品面向服务体系结构（SOA）和ESB实现混合在一起。当时是Java和EJBs取代了COBOL，成为作为企业业务应用程序的平台。这其中很多应用程序托管在大型企业服务器上，并没有实现虚拟化或其他有效的运行时隔离，让部署和操作更加复杂和危险。

None of this technology supports the kind of rapid, iterative change

and deployment that DevOps is about. Most of it is nearing end of

life, draining IT budgets into support and maintenance, and taking

resources away from new product development and technology driven

innovation. In a few cases, nobody has access to the source

code, so the systems can’t be changed at all.

这些技术都不支持DevOps所提倡的这种快速、迭代式变更已经部署。大部分技术都已接近其生命周期的末端，将IT预算消耗到支持和维护中，并将资源脱离新产品开发和技术创新。在一些情况下，没有人可以接触源代码，这样系统就根本无法更改。

Legacy technology isn’t the only drag on implementing changes.

Another factor is the overwhelming amount of data that has built up

in many different systems and different silos. Master data management

and other enterprise data architecture projects are never ending

in global banks as they try to isolate and deal with inconsistencies

and duplication in data between systems.

遗留技术并不是实现变更的唯一阻力。另一个因素是在许多不同的系统和不同的筒仓中积累了大量的数据。在全球化银行中，主数据管理和其它企业数据体系结构的项目也从未间断过，他们试图孤立和处理系统之间的不一致的地方和数据间的重复。

Dealing with Legacy Controls

Legacy controls and practices, mostly Waterfall-based and

paperwork-heavy, are another obstacle to adopting DevOps.

处理遗留控制

传统的主要是基于瀑布的繁重文书工作控制和实践，是采用DevOps的另一个障碍。

Entrenched operational risk management and governance frameworks

like CMMI, Six Sigma, ITIL, ISO standards, and the layers of

bureaucracy that support them also play a role. Operational silos are

created on purpose: to provide business units with autonomy, for

separation of control, and for operational scale. And outsourcing of

critical functions like maintenance and testing and support, with

SLAs and more bureaucracy, creates more silos and more resistance

to change.

根深蒂固的运营风险管理和治理框架，如CMMI、六西格玛、ITIL、ISO标准，以及支持它们的层层官僚机构也发挥了作用。运营筒仓创建目的：为业务单位提供自主权，分离控制，以及运营规模。此外，关键功能，如维护、测试和技术支持的外包，引入了SLA和更多的官僚机构，创造了更多的筒仓和更多的变更阻力。

DevOps initiatives need to fight against this bureaucracy and inertia,

or at least find a way to work with it.

尝试导入DevOps需要与这种官僚作风和惰性作斗争，或者至少找到一种方法来处理它。