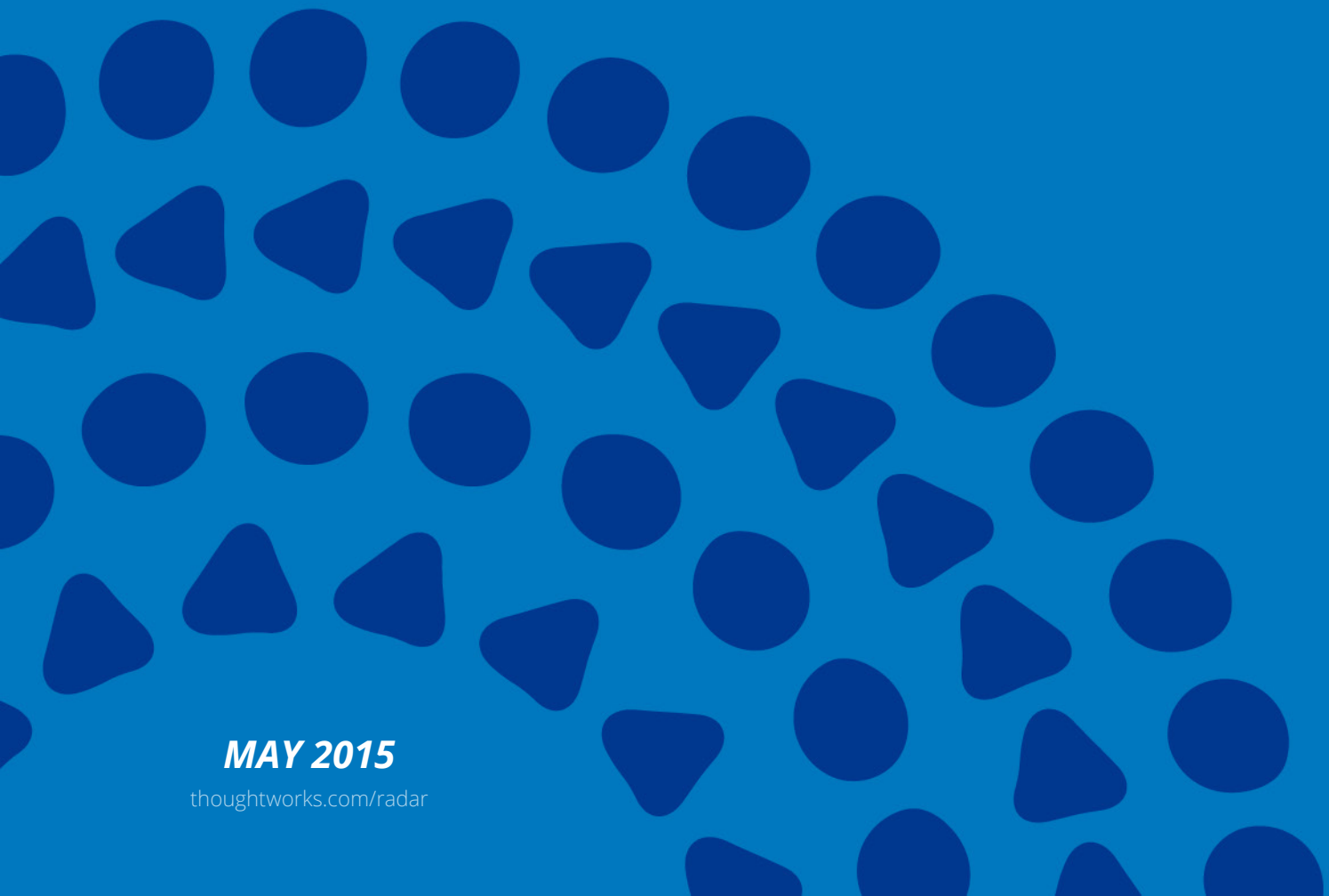
ThoughtWorks®

TECHNOLOGY RADAR

Our thoughts on the technology and trends that are shaping the future



MAY 2015

thoughtworks.com/radar

最新动态

本版精彩集锦

架构创新

组织机构已经承认了“云”将是未来的标准平台，它带来的益处和灵活性引领了软件架构的复兴。即用即扔的云基础设施开启了第一种原生的云架构：微服务。而持续交付，这种彻底改变了基于技术的业务演进方式的技术，放大了云做为架构的影响。我们期待架构级别的创新能够持续出现，像容器化和软件定义网络这类带来更多技术选择和能力的趋势能够得以延续。

微软新一波开放

尽管微软在过去对开源有所试水，包括他们对开源项目提供主机服务的平台CodePlex，但这家公司的核心资产一直是专有的，接近高度机密。而最近，微软貌似开始拥抱一种新的开放策略，在Github上发布了 .Net平台和运行时的很大一部分作为开源项目。我们希望这能为将Linux作为 .Net 的宿主平台铺平道路，让C# 能够和当前基于JVM的一堆语言来竞争。

企业中持续的安全挑战

尽管在安全和隐私方面的关注度有所增加，但上次雷达发布之后业界在这个领域并没有多大进展，我们将继续强调这个问题。开发者方面有所响应，他们增强了安全基础设施和工具，把类似 Zed Attack Prox 之类的自动化测试工具构建到部署流水线中。当然这类工具只能算整个安全方案中的一部分，我们相信所有的企业组织都需要在这个领域提高水平。

贡献者

ThoughtWorks 技术顾问委员会由以下人员组成：

Rebecca Parsons (首席技术官)	Dave Elliman	James Lewis	Rachel Laycock
Martin Fowler (首席科学家)	Erik Doernenburg	Jeff Norris	Sam Newman
Anne J Simmons	Evan Bottcher	Jonny LeRoy	Scott Shaw
Badri Janakiraman	徐昊	Mike Mason	Srihari Srinivasan
Brain Leke	Ian Cartwright	Neal Ford	Thiyagu Palanisamy
Claudia Melo			

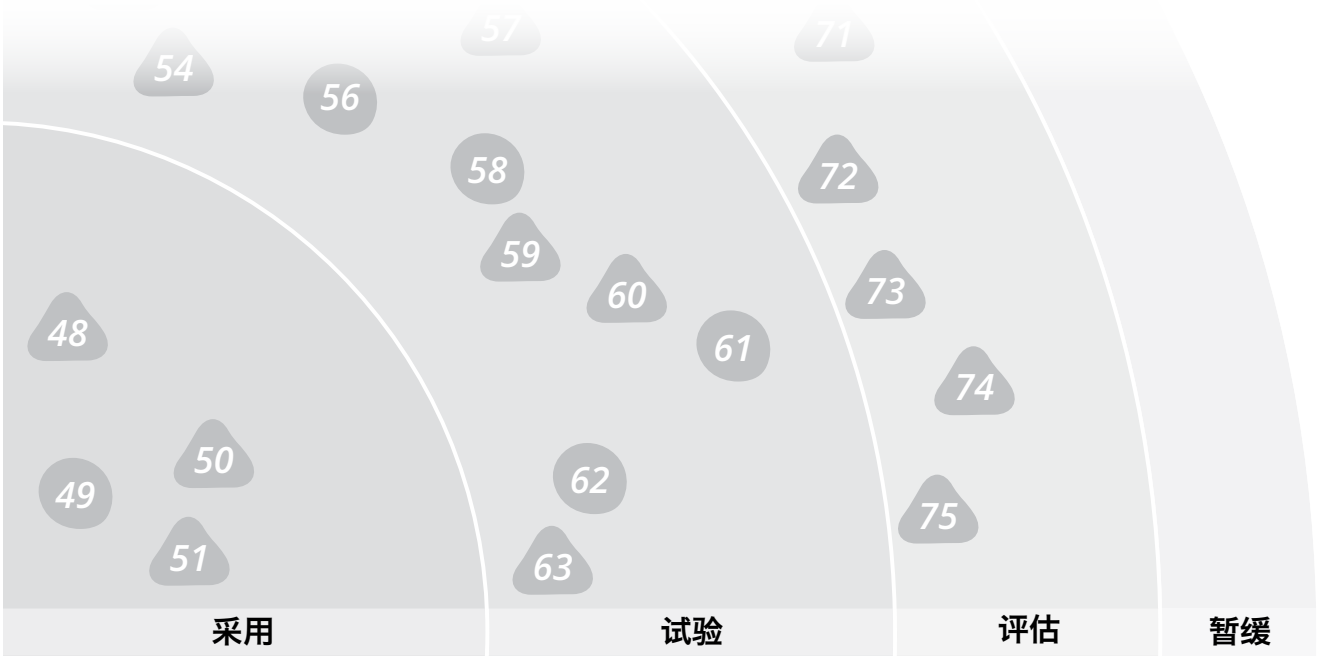
本期技术雷达中文译者：

代俊锋	付　莹	高　莉	韩　锴	何　飞
何海洋	李光磊	李　建	刘文博	刘宇峰
邱俊涛	孙建康	王　健	王晓峰	文竞成
杨　栋	银大伟	于晓强	申　章	禚娴静

关于技术雷达

ThoughtWorks人酷爱技术。我们对技术进行构建、研究、测试、开源、描写，并持之以恒地推动技术的发展——以求造福大众。我们的使命是追求卓越软件并掀起IT革命。我们创建并分享ThoughtWorks技术雷达，正是为了达成这一使命。ThoughtWorks技术顾问委员会由ThoughtWorks一群资深的技术领导者组成，他们定期召集会议讨论ThoughtWorks的全球技术战略，分析对行业产生重大影响的技术趋势，从而创建了技术雷达。

雷达以独特的形式记录技术顾问委员会的讨论结果，从CIO到开发人员，雷达为各方利益相关者提供价值。这些内容只是简要的总结，我们建议您探究这些技术以了解更多细节。雷达的本质是采用图形化方式将各种技术归类为技术、工具、平台和语言及框架四个象限。倘若雷达中的某种技术可以被归到多个象限中，我们会选择看起来最合适的一个。我们还进一步将这些技术分为四个环中，由此反映我们目前对它们持有的态度。这四个环分别为：



采用

我们强烈主张业界采用这些技术。如果适合我们的项目，我们会采用它们。

试验

值得追求。重要的是理解如何建立这种能力。企业应该在风险可控的项目中尝试该项技术。

评估

为了确认它将如何影响您所在的企业，值得作一番探究。

暂缓

谨慎推行。

自上次雷达发表以来新出现或发生显著变化的技术以三角形表示，而没有变化的技术则以圆形表示。每个象限的详细图表显示各技术发生的移动。我们感兴趣的技术实在太多，远不是如此大小的文档能合理容纳的，因此我们略去了上期雷达中已包含的许多技术，为新技术腾出空间，略去某项技术并不表示我们不再关心它。

要了解关于雷达的更多背景，请参见 thoughtworks.com/radar/faq

THE RADAR

技术

采用
1. Consumer-driven contract testing new
2. Focus on mean time to recovery
3. Generated infrastructure diagrams new
4. Structured logging

试验
5. Canary builds
6. Datensparsamkeit
7. Local storage sync
8. NoPSD
9. Offline first web applications new
10. products over projects new
11. Threat Modelling new

评估
12. Append-only data store
13. Blockchain beyond bitcoin
14. Enterprise Data Lake
15. Flux new
16. git based CMS/git for non-code new
17. Phoenix Environments new
18. Reactive Architectures new

暂缓
19. Long lived branches with Gitflow
20. Microservice envy
21. Programming in your CI/CD tool
22. SAFe™
23. Security sandwich
24. Separate DevOps team

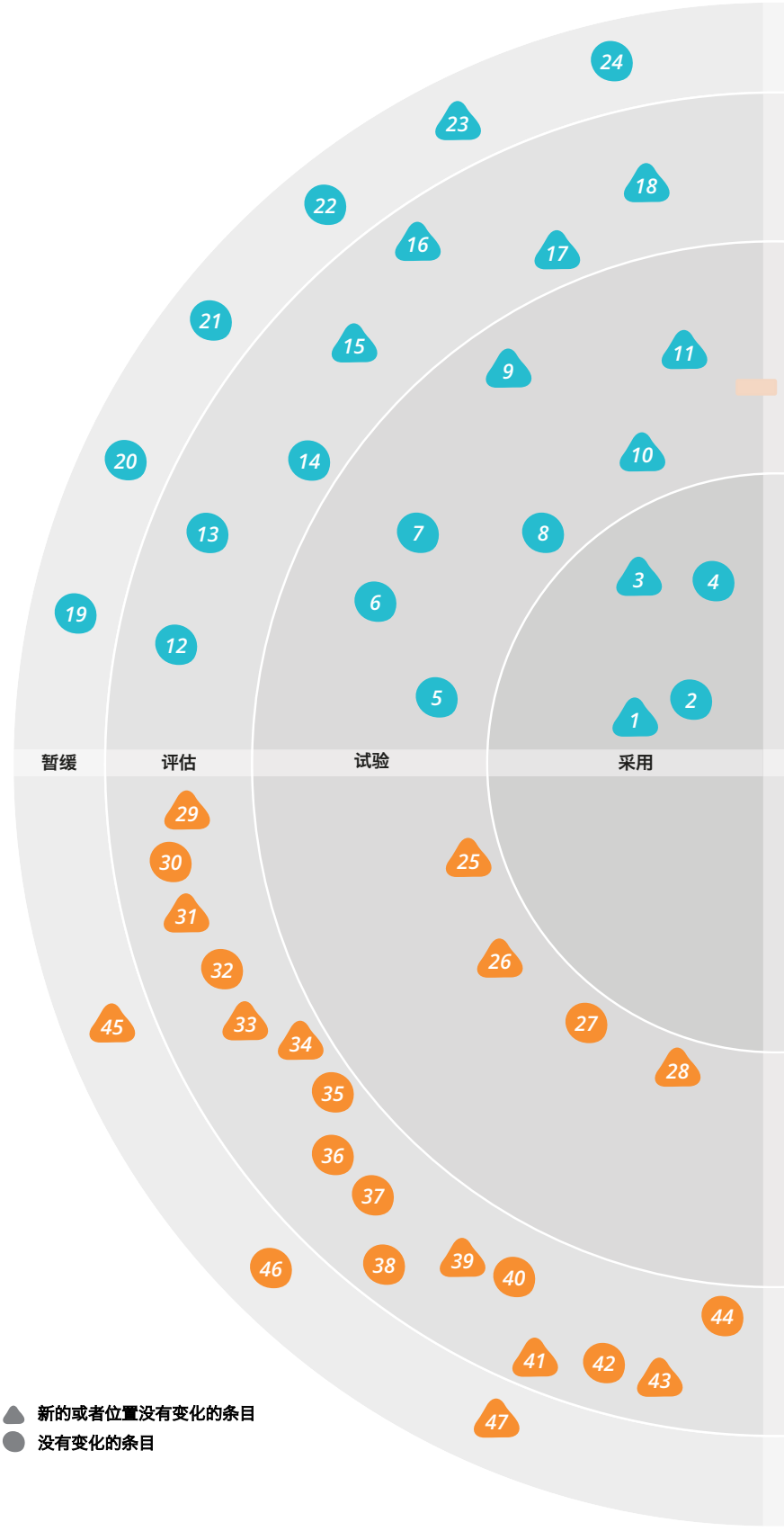
平台

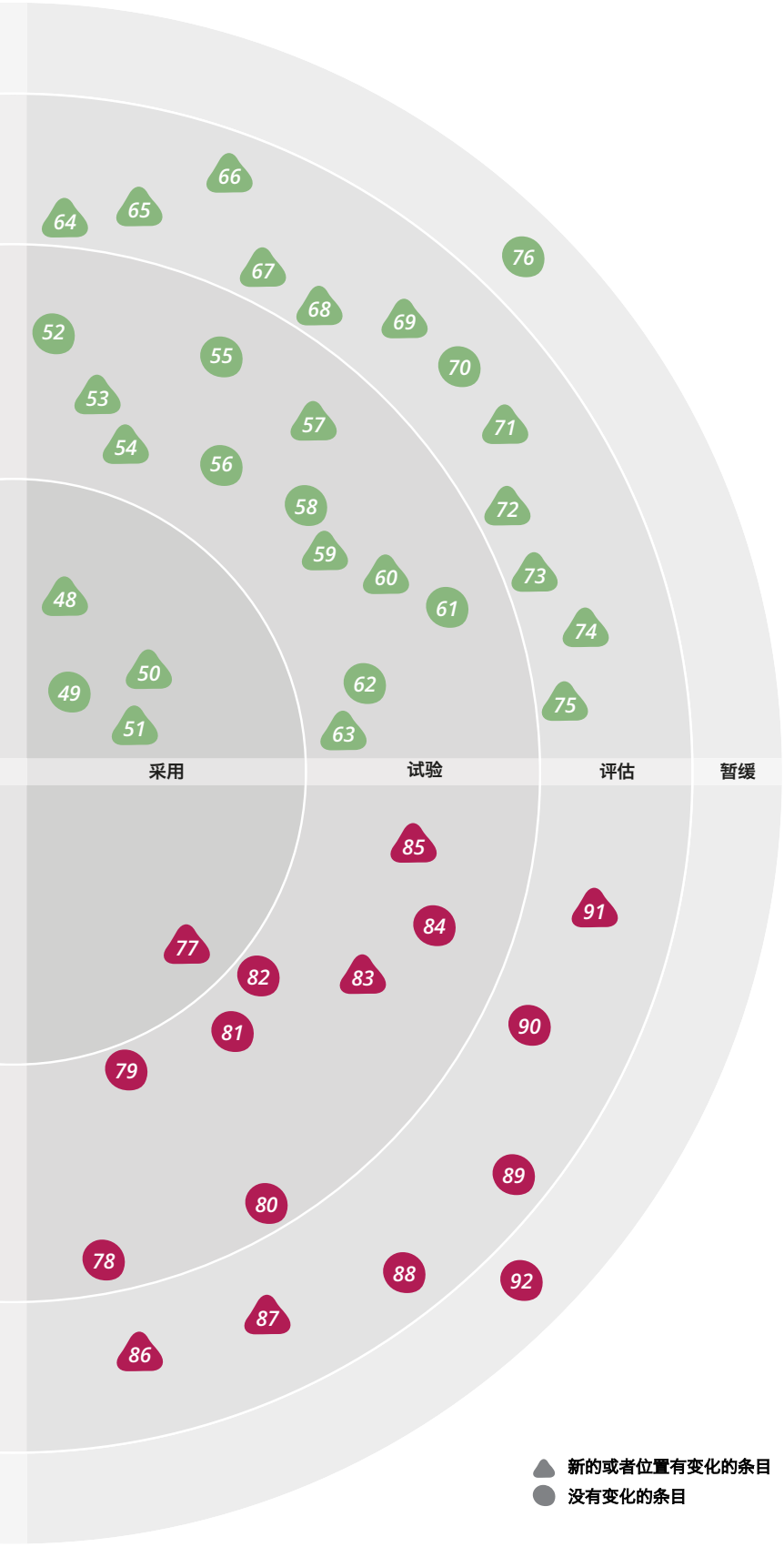
采用

试验
25. Apache Spark
26. Cloudera Impala new
27. DigitalOcean
28. TOTP Two-Factor Authentication

评估
29. Apache Kylin new
30. Apache Mesos
31. CoreCLR and CoreFX new
32. CoreOS
33. Deis new
34. H2O new
35. Jackrabbit Oak
36. Linux security modules
37. MariaDB
38. Netflix OSS Full stack
39. OpenAM
40. SDN
41. Spark Photon/Spark Electron new
42. Text it as a service / Rapidpro.io
43. Time series databases new
44. U2F

暂缓
45. Application Servers new
46. OSGi
47. SPDY new





▲ 新的或者位置有变化的条目
● 没有变化的条目

THE RADAR

工具

采用
48. Composer
49. Go CD
50. Mountebank
51. Postman

试验
52. Boot2docker
53. Brighter new
54. Consul
55. Cursive
56. Gitlab
57. Hamms new
58. IndexedDB
59. Polly new
60. REST-assured new
61. Swagger
62. Xamarin
63. ZAP new

评估
64. Apache Kafka new
65. Blackbox
66. Bokeh/Vega new
67. Gor new
68. NaCl new
69. Origami new
70. Packetbeat
71. pdfmake new
72. PlantUML new
73. Prometheus new
74. Quick new
75. Security Monkey new

暂缓
76. Citrix for development

语言和框架

采用
77. Nancy

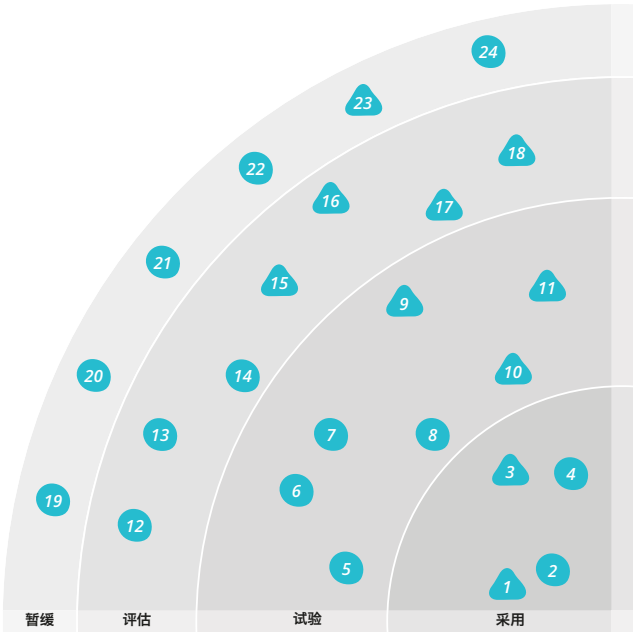
试验
78. Dashing
79. Django Rest
80. Ionic Framework
81. Nashorn
82. Om
83. React.js
84. Retrofit
85. Spring Boot

评估
86. Ember.js new
87. Flight.js
88. Haskell Hadoop library
89. Lotus
90. Reagent
91. Swift

暂缓
92. JSF

技术

当多个独立开发的服务通过API交互的时候，API提供端的改动会让它所有的消费端调用失败。消费端服务通常也不会直接去连接处于开发中的提供端服务来进行测试，因为这样的测试是缓慢且脆弱的（martinfowler.com/articles/nonDeterminism.html#RemoteServices），因此最好的方式是采用测试替身（martinfowler.com/bliki/TestDouble.html），但这又引出了测试替身和实际的API之间失去同步的风险。消费端的开发团队可以通过使用集成的合约测试（martinfowler.com/bliki/IntegrationContractTest.html）来保护自己——其比较真实服务的响应和测试替身之间的一致性。这样的合约测试是非常有价值的，而更有价值的方式是消费端把它们的测试提供给API提供端——采用消费端驱动的合约（martinfowler.com/articles/consumerDrivenContracts.html），提供端可以运行所有消费端所提供的测试来验证自己的修改是不是有可能引起问题。这种**消费者驱动的合约测试**（consumer-driven context test）是成熟的微服务测试策略（martinfowler.com/articles/microservice-testing/）中非常核心的组成部分。



当我们需要一张描述当前系统的基础设施或物理架构的图形时候，我们通常会选用自己最喜欢的工具来绘制。但是当你使用云或者其他虚拟化技术的时候，这种方式却不再适用。我们可以使用这些平台本身提供API去查询实际的基础架构环境，并使用一些简单的工具比如GraphViz（graphviz.org），一些可以输出SVG格式的工具，生成实时的，**自动化的基础架构图**（automated infrastructure diagram）。

离线优先Web应用程序（Offline first web applications）提供了基于缓存和更新机制来设计 Web 应用离线访问的能力。它的实现需要在DOM中设定一个标志来检查接入设备是否在线，离线则访问本地存储，在线则同步数据。现在所有的主流浏览器都支持离线模式，通过显示的指定 HTML 属性来使本地信息可访问，同时启动如 HTML，CSS，Javascript 或其他资源的下载和缓存。当前已经有一些工具使离线优先应用的实现变的简单，如**Hoodie**（hood.ie），**CouchDB**（couchdb.apache.org），不仅如此它们还提供与本地部署的本地存储应用的集成能力。

大多数软件开发的心智模型都是做项目，在不同的档期内进行计划、执行和交付。敏捷开发极大的挑战了这种模型，通过与开发过程同时进行的持续需求发现，代替了预先的需求确定。精益创业的技术，如观察需求的A/B测试（martinfowler.com/bliki/ObservedRequirement.html），进一步削弱了这种心态。我们认为，大多数的软件开发工作应该遵循精益企业的引领（info.thoughtworks.com/lean-enterprise-book.html），将自己定义为构建支持业务流程的产品。这样的产品并没有所谓的最终交付，更多的是一个探索如何更好的支持和优化业务流程的过程，只要业务依然有价值就会不断持续下去。基于这些理由，我们提倡企业组织依照**产品而不是项目**（products rather than projects）的思路进行思考。

采用
1. Consumer-driven contract testing
2. Focus on mean time to recovery
3. Generated infrastructure diagrams
4. Structured logging

试验
5. Canary builds
6. Datensparsamkeit
7. Local storage sync
8. NoPSD
9. Offline first web applications
10. products over projects
11. Threat Modelling

评估
12. Append-only data store
13. Blockchain beyond bitcoin
14. Enterprise Data Lake
15. Flux
16. git based CMS/git for non-code
17. Phoenix Environments
18. Reactive Architectures

暂缓
19. Long lived branches with Gitflow
20. Microservice envy
21. Programming in your CI/CD tool
22. SAFe™
23. Security sandwich
24. Separate DevOps team

技术 接上页

当前大多数开发团队都意识到编写安全软件并以负责任的方式处理用户数据的重要性。他们确实面临着陡峭的学习曲线和大量的潜在威胁，其范围从有组织的犯罪和政府的间谍活动到仅仅是为"玩笑或激怒什么人"而攻击系统的年轻人。**威胁建模**（Thread modeling）（owasp.org/index.php/Category:Threat_Modeling）是一组技术，主要从防御的角度出发，帮助理解和识别潜在的威胁。当把用户故事变为"邪恶用户故事"时，这样的做法可给予团队一个可控且高效的方法使他们的系统更加安全。

Flux（facebook.github.io/flux）是 Facebook 为其互联网应用开发所采用的一种应用架构。它通常与 **react.js** 一同被提及，Flux 基于一个单向数据流，用户或外部事件对数据存储的修改会触发数据在渲染管道中向上流动。已经有好一阵子我们都没有看到任何古老的model-view-*架构的替代者了，Flux 拥抱这种 Javascript 前端应用与多个后端服务通信的现代互联网时代。

当前，大部分开发人员习惯使用 git 来管理源代码以及协作。但是，git 还可以为其他一些情况提供基础的实现机制，比如当人们需要使用基于文本化的文档进行协作的时候（这些文档可以被很容易的合并）。通过使用基于文本的可编辑格式，我们已经看到越来越多的项目把 **git**（git-scm.com）作为一个轻量化的 **CMS** 来使用。Git 有很多强大的功能，通过使用分布式的存储模型，Git 可以支持对变化的跟踪以及寻找替代品。然而，难于大范围采用 Git 的最大原因是 git 对于非编程开发人员来说并不容易学习，而我们期望看到更多的构建在 git 核心之上的工具的出现。这类工具可以为一些特殊听众简化他们的工作流程，如作为内容的作者。我们也欢迎更多的工具支持对比和合并非文本文档。

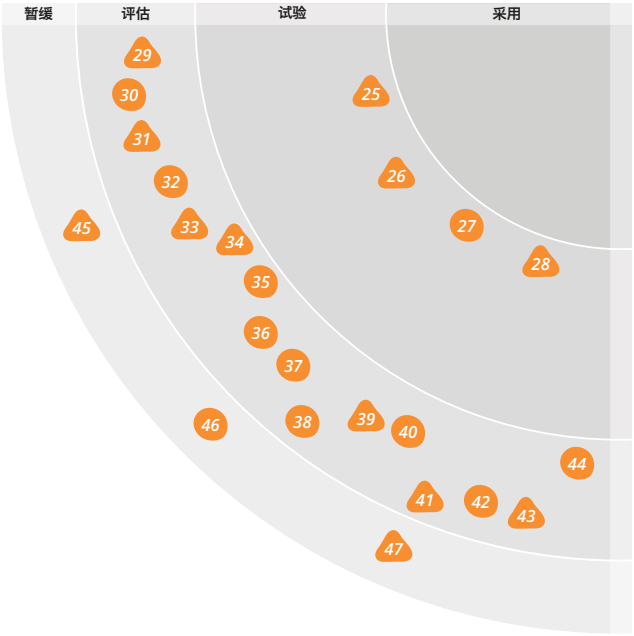
凤凰服务器（martinfowler.com/bliki/PhoenixServer.html）的想法现在已经被很好的建立并且在被应用到一些正确的问题上时带来了很多好处。而我们要部署的服务器所依赖的环境，如在某些情况下需要等待网络配置、负载均衡、防火墙端口等等情况，却已逐渐成为修改配置时的主要瓶颈和限制。**凤凰环境**的概念可以帮助这种情况。它允许我们可以用自动化的方式创建整个环境并保证定期销毁和重建整个环境的流程正确，例如在AWS上使用 **CloudFormation**。凤凰环境可以支持为测试，开发，UAT 等等配置一个全新的环境。它也可以简化对灾难恢复环境的配置。由于凤凰服务器的模式并不总是可行的，我们需要小心地对待诸如状态和依赖，在蓝绿部署(martinfowler.com/bliki/BlueGreenDeployment.html)中用它对环境配置进行重置可以成为一种方式。

函数式反应型编程在过去的几年渐渐开始流行起来，同时这个概念越来越多被延伸到在分布式系统架构中。根据**反应型编程宣言**(reactivemanifesto.org)，反应型架构（reactive architectures）主要基于通过一个网络下独立进程间的单向，异步的不可变事件流（可能具体实现为微服务）。在正确设置下，基于这种架构的系统容易扩展，具有弹性，也会减小了各个独立处理单元间的耦合。但是完全基于异步消息传递的架构同时会引入相应的复杂度，并且常常会依赖于一些专有的框架。所以我们推荐在选择这种架构风格前首先了解自己系统对于性能以及可扩展性的需求。

对于安全，传统的方式依赖于前期的需求规格以及最后阶段的验证。这种"**安全三明治**"的方式很难应用于敏捷团队，因为大部分设计都贯穿于整个过程，而且也没有持续交付所提供的自动化便利。公司或者组织应着眼于如何在整个敏捷开发周期中注入安全实践。这包括：正确评估当前威胁模型的级别以做前期设计；考虑何时将安全问题划分为独立的故事、验收标准、或全局的非功能性需求；在构建流水线中引入静态或动态的自动化安全测试；考虑如何将更深层次的测试，如渗透测试，引入到持续交付的发布过程中。正如 DevOps 的出现使得过去相互博弈的团队能够重新合作一样，同样的事情也正发生在安全人员和开发人员身上。（尽管我们并不喜欢安全三明治模型，但这也比根本不考虑安全要好得多，糟糕的是，这依然是一种非常普遍的状况。）

平台

Apache Spark（spark.apache.org）作为一种快速和通用的大规模数据处理引擎已取得稳步进展。该引擎基于Scala实现，非常适合于那些在多并行操作之间重用数据工作集的应用程序。它即可以作为一个独立集群，也可以作为Hadoop的YARN集群的一部分来工作。它可以从不同的源来访问数据，比如HDFS，Cassandra，S3等。不仅如此，Spark还提供了许多更高级的操作符，以便简化数据并行应用程序的开发。作为一种通用的数据处理平台，它使许多更高级别的工具的开发成为可能，如交互式SQL（Spark SQL），实时流媒体（Spark Streaming），机器学习库（MLib），R-on-Spark等。



一段时间以来，Hadoop社区一直在尝试把低延迟和交互式SQL查询能力带到Hadoop平台中（称为SQL-on-Hapdoop）。开源数据库引擎Cloudera Impala，Apache Drill和Facebook的Presto都在2014年应运而生。我们认为，SQL-on-Hadoop这一趋势标志着一个重要的转折，它将 Hadoop 的定位从与数据库互补的批处理，转变为某种可以与之竞争的技术。

Cloudera Impala（cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html）是早期的SQL-on-Hadoop 平台之一。它是一个基于C++的，支持大规模并行处理的分布式查询引擎。Impala 守护进程是这个平台的核心组件，其负责协调 Impala 集群中跨一个或多个节点间 SQL 查询的执行。 Impala 充分利用了 Hive 的元数据目录来共享两者的数据库和表。Impala 还提供了命令行工具以及 JDBC 和 ODBC 驱动程序供应用程序使用。

密码仍然是一种糟糕的用户认证机制。近来我们看到有公司（如Yahoo!）采用了“无密码”的解决方案——每当你需要从一个新的浏览器登录时，一个一次性验证码会被发送到你的手机来进行认证。如果你仍在使用密码认证，我们推荐您采用能够显著提高安全性的**双阶段认证**（two-factor authentication）机制。基于时间的一次性密码算法（**TOTP**）（en.wikipedia.org/wiki/Time-based_One-time_Password_Algorithm）是目前这个领域的标准，在 Google（play.google.com/store/apps/details?id=com.google.android.apps.authenticator2）和 Microsoft（windowsphone.com/en-us/store/app/authenticator/e7994dbc-2336-4950-91ba-ca22d653759b）智能手机中都这样免费的认证应用。

采用

试验
25.Apache Spark
26.Cloudera Impala
27.DigitalOcean
28.TOTP Two-Factor Authentication

评估
29.Apache Kylin
30.Apache Mesos
31.CoreCLR and CoreFX
32.CoreOS
33.Deis
34.H2O
35.Jackrabbit Oak
36.Linux security modules
37.MariaDB
38.Netflix OSS Full stack
39.OpenAM
40.SDN
41.Spark Photon/Spark Electron
42.Text it as a service / Rapidpro.io
43.Time series databases
44.U2F

暂缓
45.Application Servers
46.OSGi
47.SPDY

平台 接上页

Apache Kylin (kylin.io)，是一个来自 eBay 公司的开源数据分析解决方案，它能够在超大数据集上进行基于SQL的多维度分析（OLAP）。Kylin 旨在构建一个基于 Hadoop 的混合 OLAP (HOLAP) 解决方案，最终将能够支持 MOLAP 和 ROLAP 风格的多维分析。你可以使用 Kylin 所提供的立方体设计器来定义立方体，并启动一个离线进程来构建它们。离线进程会进行一个预连接的步骤，将事实表和维度表连接到一个扁平化的结构中。下一个是预聚合阶段，各个单独的立方体被 Map Reduce 任务会构建出来。其结果被存储在 HDFS 序列文件中，之后被载入 HBase 。数据请求可以由基于 SQL 的工具提交 SQL 产生。查询引擎（基于**Apache Calcite**）会决定目标数据集是否在 HBase 中存在。如果存在，该引擎会直接访问 HBase 中的目标数据，以次秒级延迟返回结果。如果目标数据集不存在，该引擎会将这些查询转向**Hive**（或者是集群中任何其它可以用 SQL 查询 Hadoop 的方案）。

CoreCLR (github.com/dotnet/coreclr) 和 **CoreFX** (github.com/dotnet/corefx) 是 .NET 的核心平台和框架。虽然算不上是什么新闻，他们最近被微软开源了。一个主要的变化是这些依赖是基于二进制文件来部署的，不再需要事先安装在机器上。这使得并行部署变得容易，允许应用程序可以无冲突的使用不同版本的 .NET框架。你可以安装 .NET依赖到任何环境中，基于 .NET编写代码成为了一个实现细节。从外部依赖的角度来看，一个用.NET实现的工具与用C 语言编写的东西并没有什么不同，这就使它成为编写通用应用程序和工具的一个更有吸引力的选择。CoreFX 也已被分解为独立的NuGet依赖，从而使应用能够按需使用，这让 .NET 的应用和库占的空间的更小，并使得替换部分框架更加容易。

Heroku用它的12要素应用模型改变了我们关于构建、部署、托管 Web 应用的方式。**Deis** (deis.io) 将 Heroku PaaS 模型封装到一个开源框架中，部署在可被托管在任何地方Docker容器中。Deis仍在进化当中，但对于那些符合12要素模型的应用来说，它具备大大简化部署，并在你自选的环境中进行托管的潜力。Deis也已成为Docker周边丰富的平台和工具生态系统中的又一鲜活事例。

预测分析在越来越多的产品中被使用，而且通常出现在面向最终用户的功能上。**H2O** (docs.0xdata.com) 是一套非常有意思的新开源工具包（其背后是一家创业公司），因为其易用的用户界面设计，使得预测分析对项目组更可用。同时它还集成了数据科学家最喜欢的一些工具：R 和 Python 语言，以及 Hadoop 和 Spark。H2O提供了很高的性能，并且依我们的经验，非常易于在运行时集成，特别是在基于 Java 虚拟机的平台上。

当 Oracle 决定停止对 Sun 公司的 OpenSSO（一个开源的访问管理平台）进行开发时，ForgeRock 决定接管它并将它集成进他们的 Open Identity Suite 中。现在它被命名为 **OpenAM** (forgerock.com/ products/open-identity-stack/openam)，成为了一个支撑 OpenID 连接和 SAML 2.0的可扩展的开放源码平台。不过，由于 OpenAM 历史悠久，导致它的代码库很庞大，并且文档也很难理解。希望在不久后，一个更轻量级的，对自动化部署和配置提供更好支持的替代方案将会出现。

Spark (spark.io)是基于云的互联设备全栈解决方案，**Spark Photon** 是一个带 wifi 模块的微控制器，而 **Spark electron** 是连接到移动网络的变体。Spark 操作系统为这些设备添加了 REST API 服务。这套解决方案降低了进入物联网，并构建你自己的可连接设备的门槛。

时间序列数据库（TSDB）是一种针对时间序列数据的处理做了优化的系统。它允许用户对各种以时间序列组织起来的数据库对象进行CRUD操作。同时它还可以在整个序列上执行统计计算。虽然时间序列数据库不是一个新的技术，但是我们还是在这些数据库应用中看到了一些新的热点，尤其是在物联网应用领域。在许多开源和商业平台（比如**OpenTSDB, InfluxDB, Druid, BlueFloodDB**等等）的促进下，时间序列数据库技术发展迅猛。另外还值得一提的是，其中一些数据库产品还使用了类似**Cassandra**和**HBase**的分布式数据库作为他们的底层存储引擎。

容器技术，PhoenixServer以及持续交付的崛起已经开始让我们摆脱昔日部署Web应用的方式。传统方式是我们需要构建出应用工件，然后将工件安装到应用服务器中。这种方式会导致较长的反馈周期，构建时间的变长，以及在生产环境中管理这些应用服务器的开销变大。除此之外，这些应用服务器也很难做自动化。在与我们一同工作的很多团队中，开始倾向于将 HTTP服务器嵌入到应用中。有很多可以选择的嵌入式服务器：Jetty, SimpleWeb, Webbit和Owin等。更容易做自动化，更容易做部署，对基础设施的投入也会减少，因此我们推荐在未来的项目中使用嵌入式的**应用服务器**而不是传统的应用服务器。

Google从2009年开发了一个实验性质的协议**SPDY** (chromium.org/spdy/spdy-whitepaper)，作为一个替代协议，它用于解决HTTP/1.1中的一些性能短板。新的 HTTP/2 标准协议包含了很多 SPDY 中性能优化的关键特性，Google已经宣布从2016年初就会停止在浏览器中支持SPDY。因此，如果你的应用需要SPDY中的特性，我们推荐你去尝试HTTP/2。

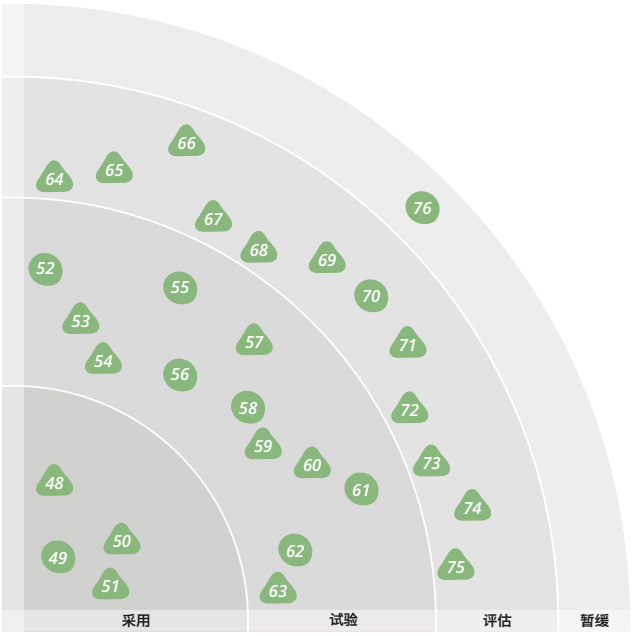
工具

尽管依赖管理的概念并不新奇，在很多技术栈下它甚至已经被作为一种基础开发实践，但在PHP社区却并非如此。**Composer**（getcomposer.org）作为 PHP 技术栈下的依赖管理工具，深受其他技术栈下依赖管理工具的影响。例如，Node 的 npm 以及 Ruby 的 Bundler 等。现如今Composer已经被PHP项目广泛使用，并且其本身也日趋成熟。虽然在对内部库的管理上，Composor还有待改进，但是对于大多数外部库的管理Composor已能够完全胜任。

在企业级应用中，对组件进行良好的测试至关重要，尤其是对于服务的分离和自动化部署这两个关系到微服务架构是否成功的关键因素，我们更需要更合适的工具对其进行测试。“服务虚拟化”这一行业术语，意指能够在组件化服务的场景下模拟特定组件的工具。**Mountebank**显然取得了不错的成绩。它是一个轻量的测试工具，可以被用于对HTTP、HTTPS、SMTP和TCP进行模拟（Mock）和打桩（Stub）。

Postman（getpostman.com/features）是一个在Chrome中使用的REST客户端插件，通过Postman，你可以创建请求并且分析服务器端返回的信息。这个工具在开发新的 API或者实现对于已有API的客户端访问代码时非常有用。Postman支持OAuth1和OAuth2，并且对于返回的JSON和XML数据都会进行排版。通过使用Postman，你可以查看你通过Postman之前发起过的请求，并且可以非常友好的编辑测试数据去测试API在不同请求下的返回。同时，虽然我们不鼓励录屏式的测试方法，但是Postman提供了一系列的拓展允许我们将它作为跑测试的工具。

Brighter（iancooper.github.io/Paramore/Brighter.html）是一个基于.Net的开源工具库，主要实现了命令调用模式。我们从正在使用它的一些团队中收到了很好的反馈，尤其在与端口模式、适配器模式和命令查询职责分离模式（**CQRS**）一起使用的时候。特别值得一提的是，它还可以很好地与**Polly**（github.com/michael-wolfenden/Polly）集成并提供熔断器模式的支持。



支持DNS和基于HTTP发现机制的服务发现工具**Consul**（consul.io）持续让我们印象深刻。它提供了定制化的注册服务健康检查并标记不健康实例的功能远胜于其他类似的工具。更多时兴的工具与Consul的集成使其功能更加强大。Consul Template (github.com/hashicorp/consultemplate) 可以直接使用Consul的信息来填充配置文件，使得像用mod_proxy进行客户端负载均衡更加容易。在使用Docker的场景里，有了registrator（github.com/gliderlabs/registrator）的帮助，只需要很小的工作量就可以自动化地向Consul注册Docker容器，使得管理基于容器技术的配置更加容易。

采用
48.Composer
49.Go CD
50.Mountebank
51.Postman

试验
52.Boot2docker
53.Brighter
54.Consul
55.Cursive
56.Gitlab
57.Hamms
58.IndexedDB
59.Polly
60.REST-assured
61.Swagger
62.Xamarin
63.ZAP

评估
64.Apache Kafka
65.Blackbox
66.Bokeh/Vega
67.Gor
68.NaCl
69.Origami
70.Packetbeat
71.pdfmake
72.PlantUML
73.Prometheus
74.Quick
75.Security Monkey

暂缓
76.Citrix for development

工具 接上页

在我们软件开发领域，盲目地假设网络总是可靠，服务器总是能够快速并正确的响应导致了许多失败的案例。**Hamms**（github.com/kevinburke/hamms）是一个有趣的开源工具，它可以模拟一个行为损坏的HTTP服务器，触发一系列的失败，包括连接失败，或者响应缓慢，或者畸形的响应。它可以帮助我们更优雅的测试我们的软件在处理异常时的反应。

我们的多个使用.NET技术栈的项目已经在推广使用**Polly**（github.com/michael-wolfenden/Polly）来帮助我构建基于微服务的系统。它鼓励使用基于流畅表达式的透明错误处理机制，以及包含了多种断路模式（Circuit Breaker Pattern），如重试，不断重试，稍后重试。在其他语言中已经存在类似的程序库，如Java中的Hystrix，而Polly是 .NET家族的一个很好补充。

REST-assured（code.google.com/p/rest-assured）是一个用于测试和验证RESTful服务的Java DSL。它使得为基于HTTP的RESTful服务编写测试变得更加简单。REST-assured支持不同类型的REST请求，并且可以验证请求从API返回的结果。它同时提供了JSON校验机制，用于验证返回的JSON数据是符合预期的。

ZED Attack Proxy (**ZAP**)（owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project）是一个OWASP的项目，允许你以自动化的方式探测已有站点的安全漏洞。可以用来做定期的安全测试，或者集成到CD的Pipleline中提供一个持续的常规安全漏洞检测。使用ZAP这样的工具并不能替换掉对安全的仔细思考或者其他的系统测试，但是作为一个保证我们的系统更安全的工具，还是很值得添加到你的工具集里。

相对于通过发送同步点对点请求的方式修改状态，最近许多企业级软件开发都在致力于基于异步不变事件序列的架构演进。**Apache Kafka**(kafka.apache.org)是一个开源消息框架，它支持基于有序的发布消息到许多独立的轻量级的消费方的架构风格。Kafka的独特设计使它能够在保持消息顺序强相关的前提下动态增加消费方的数量 。

Blackbox（github.com/StackExchange/blackbox）是一个用于加密源代码仓库中特定文件的简单工具。如果你需要存储密码或者私钥的时候，这个工具特别实用。Blackbox 可以和Git，Merurial和Subversion结合使用，并使用GPG 加密。每个用户拥有自己的密钥，使得细粒度级别的权限撤销变得很容易。这个领域正在发生很多变化，一些其他的工具也可以考虑包含进来，如 **git-crypt** 和 **Trousseau**。

在数据科学和分析的世界里，大部分工作都是使用 Python和R来完成，但是这两个语言只为Web可访问的可视化图形绘制提供了有限的几个支持。一种做法是将分析的结果转换成为能够在浏览器里很容易呈现和交互的格式。我们知道有两个工具来尝试这样做。**Bokeh** (bokeh.pydata.org)是一个可以让你创建像 D3.js 一样风格的交互式可视化的Python和JavaScript库，但是在处理大数据集或者流式的数据集时，具有更高的性能。**Vega** (trifacta.github.io/vega)是一种针对D3的声明式可视化语法，它接收服务器端生成的 JSON数据并将可视化描述转化为D3.js的代码。

Gor (github.com/buger/gor)是一个开源工具， 可以实时捕获线上HTTP请求，并在测试环境中重放这些HTTP请求，以帮助我们使用到这些产品环境数据来持续测试我们的系统。使用它之后可以大大提高我们在产品部署，配置修改或者基础架构变化时的信心。

NaCl (nacl.cr.yp.to) 库（读作'Salt'）提供了关于加密，解密和数字签名的一系列的功能，使得实现安全的网络传输，或者满足其他密码学方面的需求变得简单。尽管有一些其他的工具库也能提供这些功能，NaCl承诺提供更快的速度和更简单易用的API。当前支持C和C++的库，关于Python的封装正在进行中。

Origami (facebook.github.io/origami)是一款免费的用户原型设计工具，其对常用功能提供了大量的快捷键操作。它为将原型设计导出为代码片段提供了可能性，支持的语言有：iOS开发上的Objective-C，Android开发上的Java，以及Web开发上的Javascript。该工具可以被用来快速构建面向用户的交互式原型和测试用户使用流程。根据从一些团队收集的使用经验来看，我们建议您在需要时对该工具进行考察。

Pdfmake (github.com/bpampuch/pdfmake)是一个可以在浏览器里直接生成和打印PDF文档的JavaScript库。使用pdfmake，你可以创建一个支持表、列和富样式等结构元素的文档，再通过辅助方法创建并打印或者下载为不包含客户端JavaScript的PDF文件。

在我们的经验中，相比其他办法而言，通过在一开始创建大量详尽的设计图表来开发软件系统，并不是什么好的选择。但用图表来表达系统中格外复杂和难懂的部分却往往是个好的想法，何况UML本身也已经提供了若干实用的、众所周知的图表。我们喜欢用**PlantUML**（plantuml.sourceforge.net）来创建这些图表，因为它让我们以清晰的文字形式来表达图表背后的意图，而不用再去摆弄那些"过度"的图形化工具。同时，文字形式的表达方式还支持版本管理，并且可以和源代码存放在一起。

SoundCloud最近开源了一个Graphite的替代品：**Prometheus**（prometheus.io)。SoundCloud在解决生产环境中使用**Graphite**所遇到的困难的过程中，开发了Prometheus，它的工作方式和Graphite不同，主要体现在其对基于HTTP的拉模型的支持上（尽管它也支持和Graphite更类似的推模型）。不仅如此，它还因为支持根据获取到的度量指标进行告警的功能而比Graphite更胜一筹，所以，在你的运维工具套件中他会变得更加活跃。尽管在生产监控领域采用新技术需要谨慎行之，但早前的报告亦显示，SoundCloud对其在生产环境使用Prometheus表示满意，而且Docker也已参与到该工具后续的开发工作中。

Quick (github.com/Quick/Quick)是一个针对Swift和Objective-C的测试框架，它和用来做测试验证的**Nimble**捆绑发布。Quick主要用于Swift和Objective-c程序行为的验证。它和**rspec**和**jasmine**具有相同的语法风格，基础环境很容易建立。Quick良好的结构和类型断言使得测试异步程序更加容易。

Security Monkey (github.com/Netflix/security_monkey)是 Netflix Simian Army 工具系统中的一员，设计这套工具的初衷是为了确保系统是以有弹性的方式构建的。它不仅提供了针对AWS设置进行可配置的潜在安全漏洞评估功能，还对正在使用的 AWS 设置的变更进行监控并及时通知相应的负责团队。早于AWS的 Trusted Advisor Report (aws.amazon.com/premiumsupport/trustedadvisor)和 Cloudtrail (aws. amazon.com/cloudtrail) 可用之前开发出来的Security Monkey虽然提供的功能在某种程度上与他们有些重叠，但Security Monkey还提供了更多的功能。如果这些 AWS 的服务不太能够满足你的需求，Security Monkey值得一试。

语言和框架

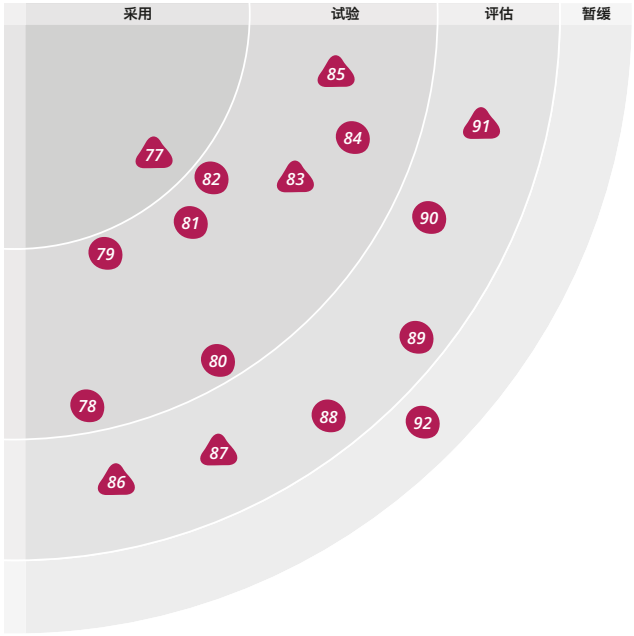
自从我们上一次在技术雷达中提到**Nancy** (nancyfx.org) 之后，它已然成为了我们在NET. 项目上的默认选择。围绕着小而垂直切片的架构和微服务需要的仅仅是这样轻量级的部署选项和不拘小节的工具。

前端 Javascript 框架持续喷井所带来的一个好处是，时不时一个新的主意出现的时候，会引起我们的思考。**React.js** (facebook.github.io/react) 是一个 UI/View 框架，在这个框架中，Javascript 函数在一个响应式的数据流中生成 HTML。我们已经见到几个小的xc项目成功的使用了 React.js，开发人员也被其干净的易组合的组件化方式所吸引。

Spring Boot (projects.spring.io/spring-boot) 能够让独立的基于Spring的应用。它非常适合启动新的微服务并且易于部署。它也通过更少的样板代码从而降低了数据访 Hibernate映射所带来的痛苦。Spring Boot 简化了基于 Spring的Java服务，这是我们所喜欢的，然而同时，我们也学会了一定要对很多依赖保持谨慎。毕竟，潜伏在表面之下的依然是Spring。

虽然并不是所有的体验都是正面的，AngularJS依然在 ThoughtWorks的项目中被广泛的使用着。我们依然建议团队评估单页Javascript应用所带来的额外复杂度对于满足需求而言，是不是真的有必要。我们也推荐评估一些其他类似的框架，在这一版的雷达中我们提出**Ember.js** (emberjs.com)，它在ThoughtWorks内部也逐渐变得流行。Ember被人称道的是，它对于惯例胜于配置非常固执己见，并且有着响应迅速的核心开发团队，其性能不错并且有一套基于ember-cli的构建工具。

在众多JavaScript框架中，我们要强调**Flight.js**（flightjs.github.io），Flight是个用于构建组件的轻量级框架。它没有通过太多"神奇"的东西来为DOM节点添加行为。它的事件驱动和基于组件的特点，促使开发人员写出低耦合的代码。这也使得测试单个组件相对容易。然而，当组件需要彼此交互的时候还是需要格外注意。Flight.js对测试的支持很少，并且容易卷入事件漩涡。继承与组合，我们喜欢组合，与之类似，我们非常喜欢Flight.js用混入的方式处理行为。



经过一些我们在真实世界中的经验，**Swift**（developer.apple.com/swift）仍然表现出非常好的前景。有些问题，如过长的编译时间，已经得到了解决。然而，持续的语言的变化会导致额外的开发工作，并使得构建你自己老版本的软件非常繁琐。测试和重构也依然痛苦。总之，虽然如此，在为苹果生态圈开发新项目的时候你还是应该考虑使用Swift。

采用
77.Nancy

试验
78.Dashing
79.Django Rest
80.Ionic Framework
81.Nashorn
82.Om
83.React.js
84.Retrofit
85.Spring Boot

评估
86.Ember.js
87.Flight.js
88.Haskell Hadoop library
89.Lotus
90.Reagent
91.Swift

暂缓
92.JSF

ThoughtWorks是一家集合了在软件咨询、交付以及产品领域富有激情并且极具前瞻性的技术工作者的软件公司。我们利用颠覆性思维帮助客户成就非凡使命，同时致力于IT产业乃至社会的变革。我们为追求卓越的软件团队提供创造性的工具。我们的产品帮助企业不断进步，不断交付满足他们重要需求的高质量软件。ThoughtWorks已经从20多年前一个芝加哥的小团队，成长为现在拥有超过3000人，分布于全球12个国家，拥有30间办公室的全球企业。这12个国家是：澳大利亚、巴西、加拿大、中国、厄瓜多尔、德国、印度、新加坡、南非、乌干达、英国和美国。

ThoughtWorks®