ThoughtWorks®

TECHNOLOGY RADAR *NOV '16*

Nossas ideias sobre tecnologias e tendências que estão moldando o futuro

thoughtworks.com/radar

#TWTechRadar

O QUE HÁ DE NOVO?

Aqui estão os assuntos em destaque nessa edição::

"DOCKER COMO PROCESSO, PAAS COMO MÁQUINA, ARQUITETURA DE MICROSSERVIÇOS COMO MODELO DE PROGRAMAÇÃO"

The microservices style of architecture highlights rising abstractions in the developer world because of containerization and the emphasis on low coupling, offering a high level of operational isolation. Developers can think of a container as a self-contained process and the PaaS as the common deployment target, using the microservices architecture as the common style. Decoupling the architecture allows the the same for teams, cutting down on coordination cost among silos. Its attractiveness to both developers and DevOps has made this the de facto standard for new development in many organizations.

O PODER DA INTELIGÊNCIA

Tópicos de longa data de P&D como aprendizagem de máquina e inteligência artificial agora têm aplicações práticas por meio de frameworks como Nuance Mix e TensorFlow. É possível baixar frameworks que vão de PLN a bibliotecas de aprendizagem de máquina. Observamos com satisfação que as empresas frequentemente abrem os códigos de bibliotecas e ferramentas sofisticadas nesse campo, as quais seriam exageradamente caras e, portanto, restritas uma década atrás, disponibilizando-as para um grande público desenvolvedor. Muitos fatores evoluíram e se somaram para tornar possíveis novas ferramentas: computação de commodity, foco em hardwares específicos como GPUs e recursos de nuvem. Talvez o seu armazenamento de Big Data esteja começando a valer a pena...

O EFEITO HOLÍSTICO DA ESTRUTURA DOS TIMES

A estrutura dos times sempre teve um grande impacto sobre uma ampla variedade de questões do desenvolvimento de software e tornou-se uma área de foco crescente diante de recursos como PaaS self-service e microsserviços. As empresas agora valorizam pensar no produto mais que projetos; empresas de tecnologia estão popularizando o estilo "você constrói, você executa" de autonomia de times, e estamos vendo a mesma filosofia de pensar no produto aplicada a projetos corporativos. Quando reestruturar times traz melhores resultados, isso mostra mais uma vez que o desenvolvimento de software ainda é majoritariamente um problema de comunicação. Construir times multifuncionais potencializa os benefícios da comunicação entre papéis de trabalho tradicionalmente segregados, o que por sua vez elimina o atrito imposto por estruturas artificiais como silos.

REALIDADE AUMENTADA E VIRTUAL CAMINHANDO PARA A POPULARIZAÇÃO

Temos visto realidade aumentada e realidade virtual (RA/RV) gerando interesse comercial, ambas tecnologias que já foram relegadas apenas a jogos e inovação. Enquanto perseguir desenhos animados virtuais trouxe a RA à atenção do público por meio de SDKs móveis, hardwares como o Oculus Rift, HTC Vive e Microsoft HoloLens estão amadurecendo a tal ponto que as primeiras a adotar podem obter benefícios sem se prejudicarem com tecnologia imatura. Apesar de plataformas de software como OpenVR e Unity estarem maduras há um bom tempo, ferramentas novas de processamento de linguagem natural (PLN) como Nuance Mix e hardware que fornece interações naturais terão um enorme impacto sobre a adoção de RA e RV. Já estamos mantendo laboratórios de RA e RV em nossos escritórios para explorar aplicações futuras como interações remotas ou sinalização de localização para varejo. Nossos experimentos demonstram a RV como um meio surpreendentemente poderoso para colaboração remota e storytelling mais compreensivas devido à sua capacidade de contornar abstração e imergir usuários diretamente em uma experiência. No entanto, observaremos desafios significativos na criação e entrega de conteúdo de RA e RV já que as habilidades e competências estão atrasadas em relação ao ritmo do hardware, especialmente nas empresas.

COLABORADORES

O Technology Radar é preparado pelo Conselho Consultivo de Tecnologia da ThoughtWorks, composto por:

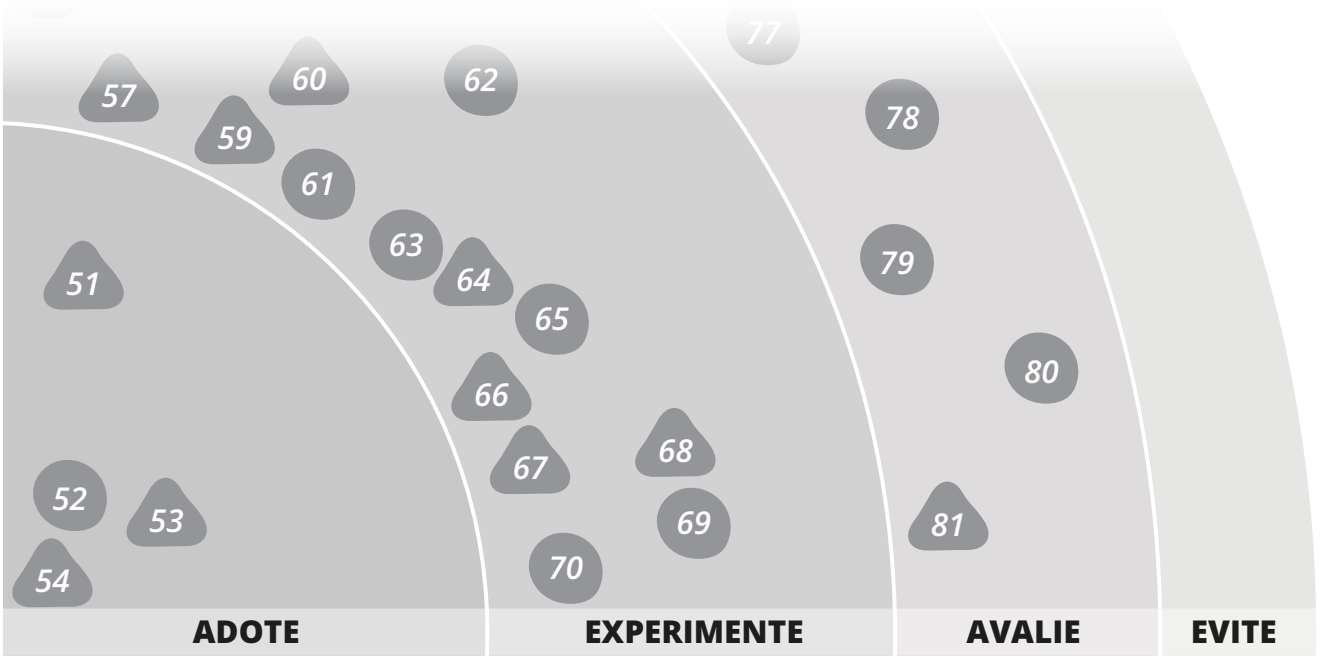
Rebecca Parsons (CTO)
Martin Fowler (Chief Scientist)
Anne J Simmons
Badri Janakiraman
Bharani Subramaniam
Camilla Falconi Crispim
Erik Doernenburg
Evan Bottcher
Fausto de la Torre
Hao Xu
Ian Cartwright
James Lewis
Jiaxing Chen
Jonny LeRoy
Marco Valtas
Mike Mason
Neal Ford
Rachel Laycock
Scott Shaw
Srihari Srinivasan
Zhamak Dehghani

TRADUÇÃO E REVISÃO: Alexandre Barbosa, Alexey Boas, Gregório Melo, Glauco Vinicius, Marco Valtas e Paula Ribas.

SOBRE O TECHNOLOGY RADAR

'ThoughtWorkers' são pessoas apaixonadas por tecnologia. Nós desenvolvemos, pesquisamos, testamos, contribuímos com código livre, escrevemos e visamos a sua constante melhoria — para todas as pessoas. Nossa missão é liderar e promover a excelência de software e revolucionar a indústria de TI. Nós criamos e compartilhamos o Technology Radar da ThoughtWorks para apoiar essa missão. O Conselho Consultivo de Tecnologia (Technology Advisory Board, ou TAB), um grupo de líderes experientes em tecnologia da ThoughtWorks, é responsável por criar o radar. O grupo se reúne regularmente para discutir a estratégia global de tecnologia para a empresa e as tendências tecnológicas que impactam significativamente a nossa indústria.

O radar captura o resultado das discussões do TAB em um formato que procura oferecer valor a uma ampla gama de pessoas interessadas, de CIOs a pessoas que desenvolvem. O conteúdo é concebido para ser um resumo conciso. Nós encorajamos você a explorar essas tecnologias para obter mais detalhes. O radar é gráfico por natureza, agrupando os itens em técnicas, ferramentas, plataformas e linguagens & frameworks. Quando itens do radar puderem ser classificados em mais de um quadrante, escolhemos aquele que parece mais adequado. Além disso, agrupamos esses itens em quatro anéis para refletir a nossa posição atual dentro deles:



Acreditamos firmemente que a indústria deveria adotar esses itens. Nós os usamos quando são apropriados em nossos projetos.

Vale a pena ir atrás. É importante entender como desenvolver essa capacidade. As empresas devem experimentar esta tecnologia em um projeto que possa lidar com o risco.

Vale a pena explorar com o objetivo de compreender como isso afetará sua empresa.

Prossiga com cautela.

Itens novos ou que sofreram alterações significativas desde o último radar são representados como triângulos, enquanto os itens que não mudaram são representados como círculos. Os gráficos detalhados de cada quadrante mostram o movimento tomado pelos itens. Temos interesse em muito mais itens do que seria razoável adicionar em um documento deste tamanho, por isso removemos muitos itens do último radar para abrir espaço para novos itens. Quando apagamos um item não significa que deixamos de nos preocupar com ele.

Para mais informações sobre o radar, veja thoughtworks.com/radar/faq

O RADAR

TÉCNICAS

ADOTE
1. Testes de contrato guiados pelo consumidor
2. Pipelines como código novo
3. Modelagem de ameaças

EXPERIMENTE
4. APIs como produto novo
5. Recompensas por bugs
6. Lago de Dados
7. Hospedando dados PII na União Europeia
8. Registros leves de decisões arquiteturais novo
9. Arquiteturas reativas
10. Arquitetura sem servidor

AVALIE
11. Consulta dirigida ao cliente novo
12. Varredura de segurança de contêineres novo
13. Políticas de Segurança de Conteúdo
14. Privacidade diferencial novo
15. Micro front-ends novo
16. ASVS do OWASP
17. Unikernels
18. Realidade virtual além dos jogos

EVITE
19. Instância única de CI para todos os times
20. REST anêmico novo
21. Inveja de Big Data
22. Subir e migrar para nuvem

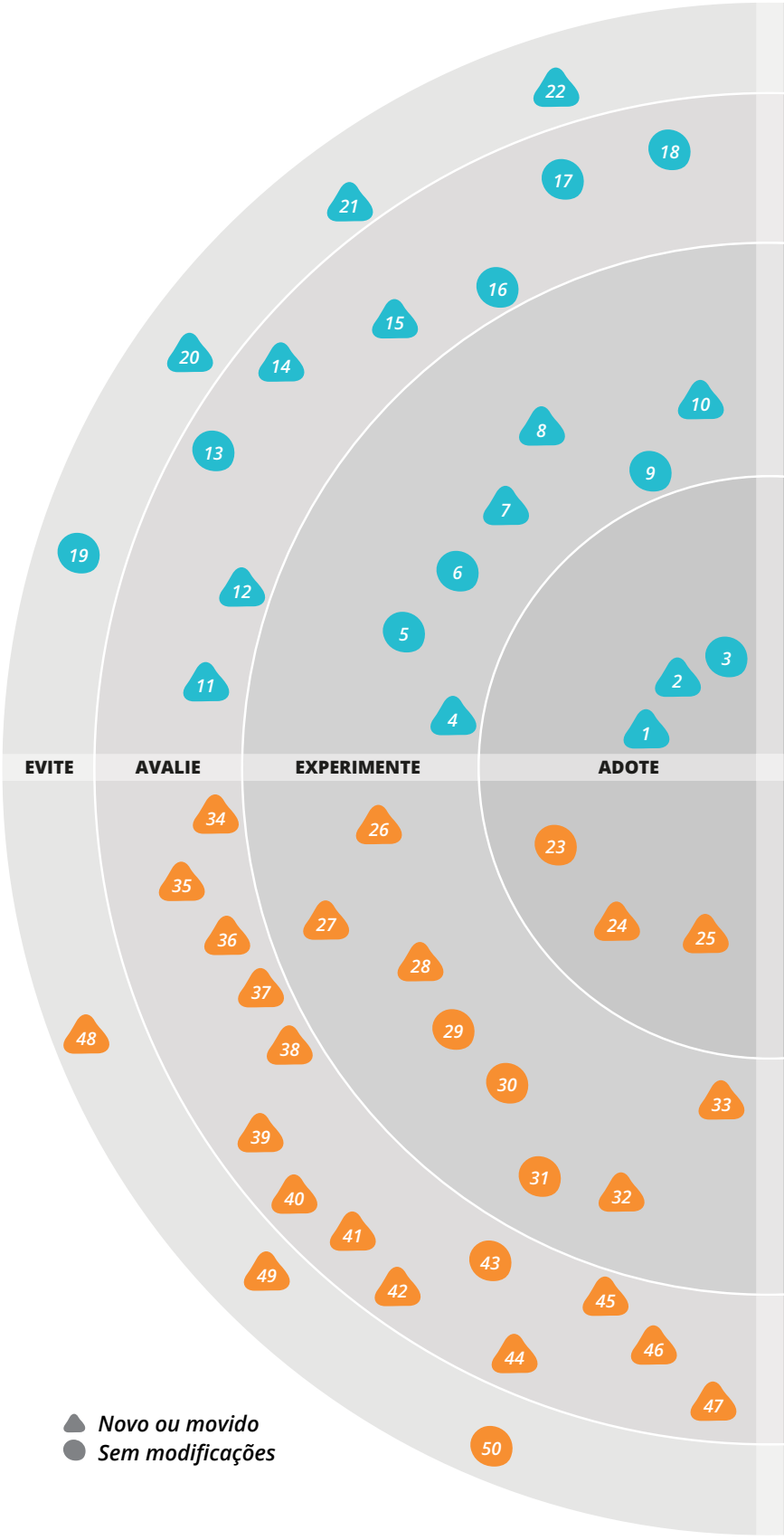
PLATAFORMAS

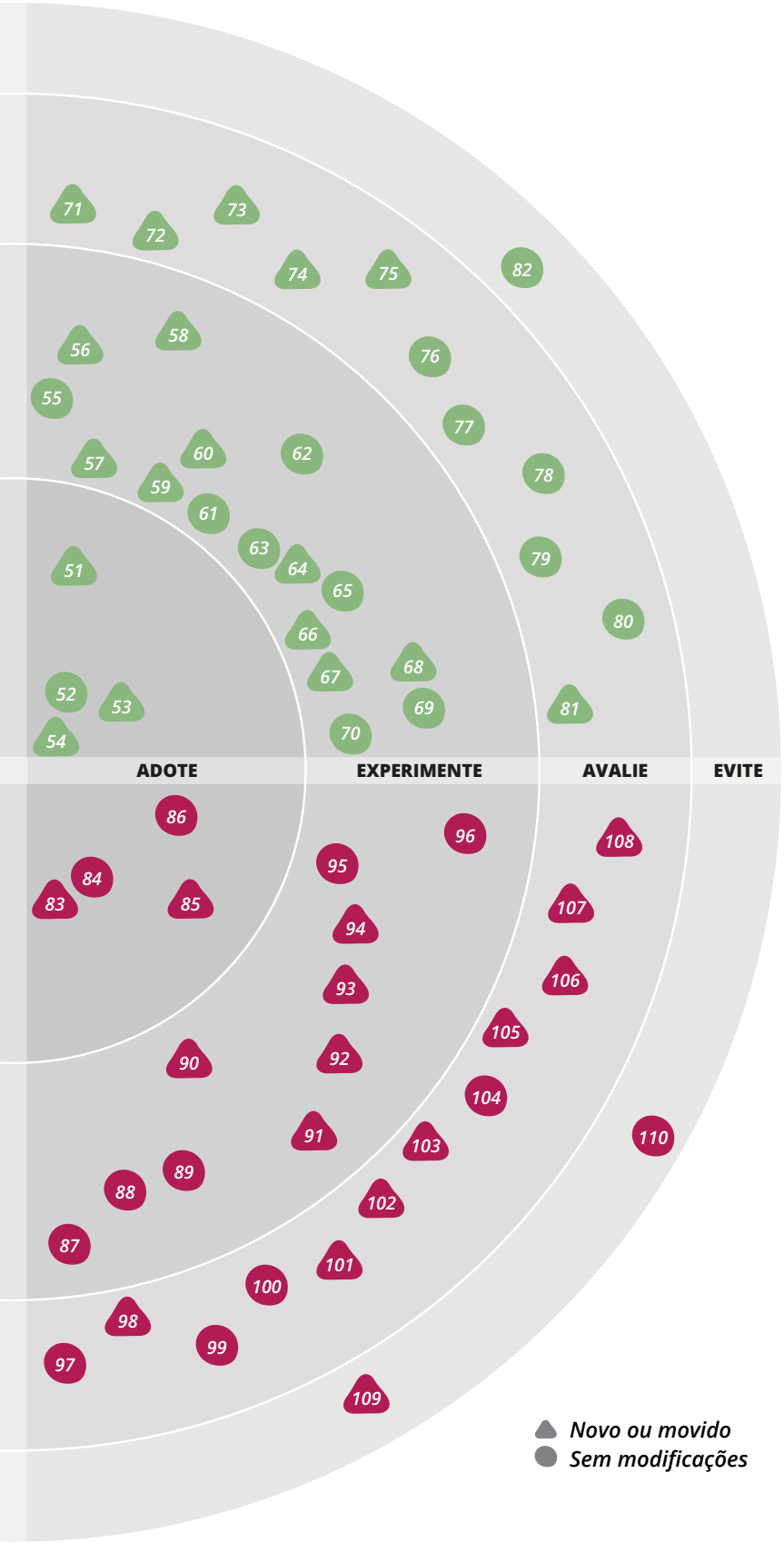
ADOTE
23. Docker
24. HSTS
25. Módulos de segurança do Linux

EXPERIMENTE
26. Apache Mesos
27. Auth0 novo
28. AWS Lambda
29. Kubernetes
30. Pivotal Cloud Foundry
31. Rancher
32. Realm
33. Unity além dos jogos novo

AVALIE
34. .NET Core
35. Amazon API Gateway
36. Apache Flink
37. AWS Application Load Balancer novo
38. Cuidado com Cassandra novo
39. Electron novo
40. Ethereum novo
41. HoloLens novo
42. IndiaStack novo
43. Nomad
44. Nuance Mix novo
45. OpenVR novo
46. Tarantool novo
47. wit.ai novo

EVITE
48. CMS como plataforma
49. API Gateways excessivamente ambiciosos
50. Nuvem privada superficial





O RADAR

FERRAMENTAS

ADOTE

51. Babel novo
52. Consul
53. Grafana novo
54. Packer

EXPERIMENTE

55. Apache Kafka
56. Espresso
57. fastlane novo
58. Galen novo
59. HashiCorp Vault
60. JSONassert novo
61. Let's Encrypt
62. Load Impact
63. OWASP Dependency-Check
64. Pa11y novo
65. Serverspec
66. Talisman novo
67. Terraform
68. tmate novo
69. Webpack
70. Zipkin

AVALIE

71. Android-x86 novo
72. axios novo
73. Bottled Water novo
74. Clojure.spec novo
75. FBSnapshotTestcase novo
76. Grasp
77. LambdaCD
78. Pinpoint
79. Pitest
80. Repsheet
81. Scikit-learn novo

EVITE

82. Jenkins como uma pipeline de implantação

LINGUAGENS E FRAMEWORKS

ADOTE

83. Ember.js
84. React.js
85. Redux
86. Spring Boot

EXPERIMENTE

87. Butterknife
88. Dagger
89. Dapper
90. Elixir
91. Enzyme novo
92. Immutable.js
93. Phoenix novo
94. Quick e Nimble novo
95. React Native
96. Robolectric

AVALIE

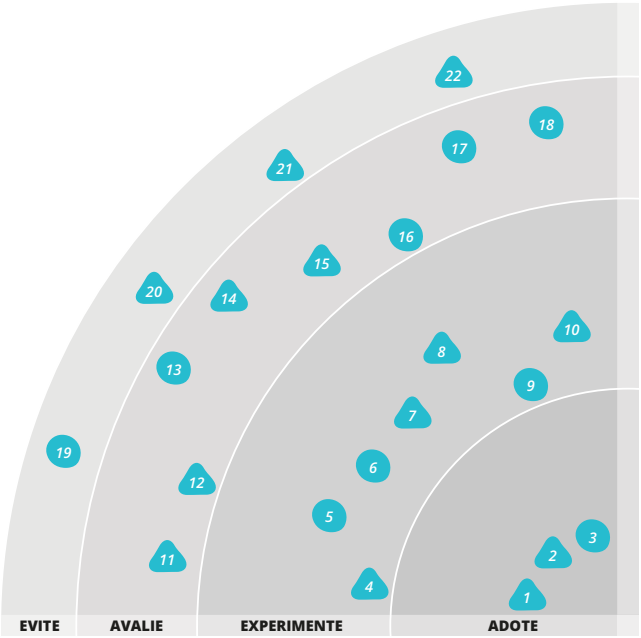
97. Aurelia
98. ECMAScript 2017 novo
99. Elm
100. GraphQL
101. JuMP novo
102. Web Física novo
103. Rapidoid novo
104. Recharts
105. ReSwift novo
106. Three.js novo
107. Vue.js novo
108. WebRTC novo

EVITE

109. AngularJS
110. JSPatch

TÉCNICAS

Decidimos trazer os **testes de contrato guiados pelo consumidor** de volta do arquivo para essa edição, ainda que tenhamos permitido que eles desvanecessem no passado. O conceito não é novo, mas com a aceitação popular dos microsserviços, precisamos lembrar as pessoas que contratos guiados pelo consumidor são parte essencial de um portfólio maduro de testes de microsserviços , permitindo implantações de serviços independentes. Mas além disso, queremos destacar que testes de contrato guiados pelo consumidor são uma técnica e uma ação que não exigem nenhuma ferramenta especial para serem implementadas. Adoramos frameworks como Pact porque eles tornam os testes de contrato apropriados mais fáceis de implementar em determinados contextos. Entretanto, notamos uma tendência dos times de se concentrarem no framework ao invés da prática geral. Escrever testes com Pact não garante que você esteja criando contratos guiados pelo consumidor, assim como em muitas situações você deve estar criando bons contratos guiados pelo consumidor mesmo quando não existe nenhuma ferramenta de testes pré-concebida.



Os times têm se dedicado à automação entre seus ambientes, incluindo sua infraestrutura de desenvolvimento. **Pipelines como código** significa definir a pipeline de implantação por meio de código ao invés de configurar uma ferramenta de CI/CD pré-existente. Lambda, Drone, GoCD e Concourse são exemplos que permitem o uso dessa técnica. Além disso, ferramentas de automação de configuração para sistemas de CI/CD como GoMatic podem ser usadas para tratar a pipeline de implantação como código—versionado e testado.

As empresas têm adotado integralmente as APIs como uma forma de expor competências de negócio para pessoas que desenvolvem, tanto externamente quanto internamente. As APIs prometem a capacidade de experimentar rapidamente novas ideias de negócio recombinando competências centrais. Mas o que diferencia uma API de um serviço de integração corporativa comum? Uma das diferenças está no tratamento de **APIs como produto**, mesmo quando o consumidor é um sistema interno. Times que constroem APIs devem entender as necessidades de seus clientes e tornar o produto atraente para eles. Os produtos também são melhorados, mantidos e suportados por eles a longo prazo. Deve haver uma pessoa proprietária para defender os interesses do cliente e se esforçar para que haja melhoria contínua. Os produtos devem ser ativamente mantidos e suportados, fáceis de encontrar e fáceis de usar. Em nossa experiência, a orientação ao produto é o ingrediente que falta para diferenciar uma integração corporativa comum de um negócio ágil construído com base em uma plataforma de APIs.

Em vários países do mundo, vemos agências governamentais buscando amplo acesso a informações pessoalmente identificáveis (personally identifiable information ou PII) e privadas. O crescente uso de soluções públicas de nuvem dificulta para as organizações proteger os dados a elas confiados por seus usuários e ao mesmo tempo respeitar todas as leis vigentes. A União Europeia possui algumas das mais progressistas leis de privacidade, e todos os grandes provedores de serviços de nuvem—Amazon, Google and Microsoft—oferecem

ADOTE
1. Testes de contrato guiados pelo consumidor
2. Pipelines como código
3. Modelagem de ameaças

EXPERIMENTE
4. APIs como produto
5. Recompensas por bugs
6. Lago de Dados
7. Hospedando dados PII na União Europeia
8. Registros leves de decisões arquiteturais
9. Arquiteturas reativas
10. Arquitetura sem servidor

AVALIE
11. Consulta dirigida ao cliente
12. Varredura de segurança de contêineres
13. Políticas de Segurança de Conteúdo
14. Privacidade diferencial
15. Micro front-ends
16. ASVS do OWASP
17. Unikernels
18. Realidade virtual além dos jogos

EVITE
19. Instância única de CI para todos os times
20. REST anêmico
21. Inveja de Big Data
22. Subir e migrar para nuvem

TÉCNICAS *continuação*

múltiplos centros de processamento de dados e regiões na União Europeia. Portanto, recomendamos que as empresas, principalmente as que têm bases de usuários globais, avaliem a viabilidade de um refúgio seguro para os dados de seus usuários **hospedando dados PII na União Europeia**. Depois que escrevemos sobre essa técnica no último Radar, lançamos um novo sistema interno que lida com informações sensíveis de todas as pessoas funcionárias da empresa, e optamos por hospedá-lo em um centro de dados localizado na União Europeia.

Embora muita documentação possa ser substituída por código e testes altamente legíveis, em um universo de arquitetura evolutiva é importante registrar determinadas decisões de design para o proveito de futuros membros do time e para supervisão externa. Os **registros leves de decisões arquiteturais** são uma técnica para capturar decisões de arquitetura importantes juntamente com seu contexto e suas consequências. Embora esses itens sejam frequentemente armazenados em uma wiki ou em uma ferramenta colaborativa, geralmente preferimos armazená-los junto com o código no sistema de controle de versões com marcação simples.

A **arquitetura sem servidor** é uma abordagem que substitui máquinas virtuais de longa duração por poder computacional efêmero, que passa a existir sob solicitação e desaparece imediatamente apos o uso. Desde o último Radar, vários dos nossos times colocaram aplicações em produção usando um estilo "sem servidor". Nossos times gostam da abordagem, ela tem funcionado bem para eles e a consideramos uma escolha de arquitetura válida. Vale notar que a arquitetura sem servidor não precisa ser uma abordagem tudo ou nada: alguns dos nossos times implantaram pedaços novos de seus sistemas usando arquitetura sem servidor e ao mesmo mantiveram uma abordagem de arquitetura tradicional para outras partes.

Embora muitos problemas que as pessoas encontram com abordagens RESTful para APIs possam ser atribuídos ao antipadrão REST anêmico, alguns casos de uso justificam a exploração de outras abordagens. Particularmente, organizações que precisam dar suporte a uma longa cauda de aplicações de clientes (e, portanto, uma provável proliferação de versões de API, ainda que elas usem contratos guiados pelo consumidor)—e tenham uma grande parcela de suas APIs dando suporte ao estilo lista interminável em feeds de atividade—podem atingir alguns limites em arquiteturas RESTful. Esses limites podem ser ocasionalmente mitigados por meio do uso da abordagem **consulta dirigida ao cliente** para interação cliente-servidor. Temos visto essa abordagem ser bem sucedida tanto com GraphQL quanto com Falcor, nos quais clientes têm maior controle sobre o conteúdo e a granularidade dos dados devolvidos a eles. Isso aumenta sim a responsabilidade da camada de serviço e ainda pode levar a um forte acoplamento com o modelo de dados subjacente, mas os benefícios podem valer a exploração caso APIs RESTful bem modeladas não estejam funcionando para você.

A revolução dos contêineres instigada pelo Docker reduziu massivamente o atrito ao mover aplicações entre ambientes, mas ao mesmo tempo colocou em cheque os tradicionais controles sobre o que pode ir para produção. A técnica de **varredura de segurança de contêineres** é uma resposta necessária a esse vetor de ameaça. O Docker agora fornece suas próprias ferramentas de varredura de segurança, a exemplo do CoreOS, e também obtivemos sucesso com os Benchmarks de Segurança do CIS. Para qualquer abordagem que você escolha, acreditamos que o tópico da validação da segurança de contêineres automatizada é de grande valor e necessária quando pensamos em PaaS.

É de conhecimento geral há algum tempo que conjuntos de dados anônimos em massa podem revelar informações sobre indivíduos, especialmente quando múltiplos conjuntos de dados são cruzados entre si. Com a crescente preocupação com os dados pessoais, algumas empresas— incluindo Apple e Google— estão recorrendo a técnicas de **privacidade diferencial** para aprimorar a privacidade individual e ao mesmo tempo manter a capacidade de realizar análises úteis em grandes quantidades de usuários. Privacidade diferencial é uma técnica de criptografia que procura maximizar a precisão de consultas estatísticas de um banco de dados, ao mesmo tempo minimizando as chances de identificação de seus registros. Esses resultados podem ser atingidos por meio da introdução de uma pequena quantidade de "ruído" nos dados, mas é importante notar que é uma área de pesquisa em andamento. A Apple anunciou planos de incorporar privacidade diferencial a seus produtos—e nós genuinamente aplaudimos o comprometimento da empresa com a privacidade de seus clientes—mas o tradicional sigilo da Apple deixou dúvidas em algumas pessoas especialistas em segurança. Continuamos recomendando Datensparsamkeit como abordagem alternativa: simplesmente armazenar a quantidade mínima de dados que você de fato precisa vai garantir resultados melhores de privacidade na maioria dos casos.

TÉCNICAS *continuação*

Temos visto benefícios significativos com a introdução de arquiteturas de microsserviços, o que permitiu aos times escalar a entrega de serviços implantados e mantidos de maneira independente. Entretanto, os times têm frequentemente encontrado dificuldades para evitar a criação de monólitos de front-end—grandes aplicações de navegador que estão se alastrando e que são difíceis de manter e evoluir, assim como as aplicações monolíticas do lado do servidor que abandonamos. Temos visto emergir uma abordagem que nossos times chamam de **micro front-ends**. Nessa abordagem, uma aplicação web é dividida por suas páginas e funcionalidades, com cada funcionalidade sendo assumida de ponta a ponta por um único time. Existem múltiplas técnicas para unir as funcionalidades da aplicação—algumas velhas e algumas novas—em uma experiência de usuário coesa, mas o objetivo continua sendo permitir que cada funcionalidade seja desenvolvida, testada e implantada de forma independente das demais. A abordagem BFF - back-end para front-ends funciona bem aqui, com cada time desenvolvendo um BFF para dar suporte a seu conjunto de funcionalidades da aplicação.

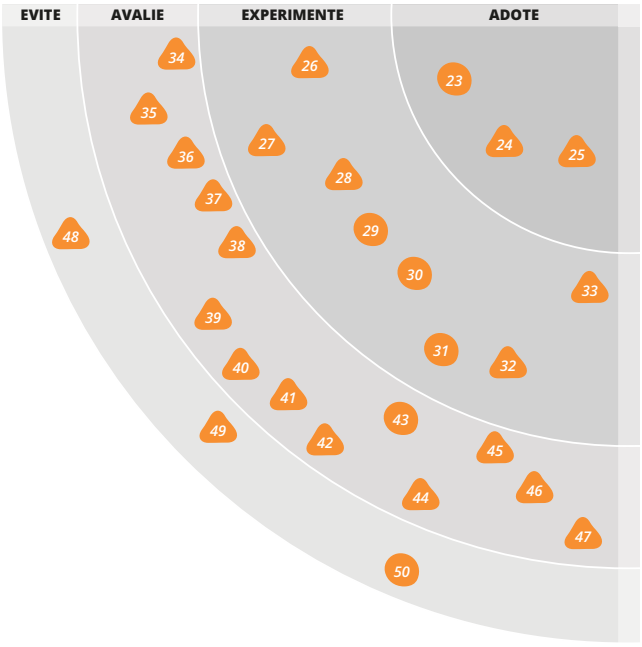
Com a crescente popularidade do padrão BFF - back-end para front-ends e do uso de frameworks ligação de dados em sentido único como React.js, notamos uma reação negativa às arquiteturas REST. As pessoas que criticam acusam o REST de provocar interações ineficientes e verbosas entre sistemas e de não se adaptar à medida que as necessidades do cliente evoluem. Elas oferecem frameworks como GraphQL ou Falcor como mecanismos de busca de dados alternativos que permitem ao cliente especificar o formato dos dados retornados. Mas em nossa experiência, o REST não é a causa desses problemas. Ao contrário disso, eles acontecem quando se deixa de modelar adequadamente o domínio como um conjunto de recursos. Desenvolver ingenuamente serviços que simplesmente expõe modelos de dados hierárquicos estáticos por meio de URLs com templates resultam em uma implementação de **REST anêmico**. Em um domínio ricamente modelado, o REST deve habilitar mais que uma simples busca repetitiva de dados. Em uma arquitetura RESTful completamente evoluída, eventos de negócio e conceitos abstratos também são modelados como recursos, e a implementação deve fazer uso efetivo de hipertexto, relações de link e tipos de mídia para maximizar o desacoplamento entre serviços. Esse antipadrão tem relação direta com o padrão do Modelo de Domínio Anêmico e resulta em serviços com baixas classificações no Modelo de Maturidade de Richardson. Temos mais sugestões para a concepção de APIs REST efetivas nesse artigo do Insights.

Continuamos vendo organizações perseguindo tecnologias "da moda", assumindo complexidade e risco desnecessários quando uma escolha mais simples funcionaria melhor. Um tema específico é o uso de sistemas distribuídos de Big Data para conjuntos relativamente pequenos de dados. Esse comportamento nos leva a trazer a **inveja de Big Data** de volta ao anel Evite, com alguns pontos adicionais de nossas experiências recentes com dados. A base de dados Apache Cassandra promete sólida escalabilidade em hardware commodity, mas temos visto times sobrecarregados por sua complexidade arquitetural e operacional. A menos que você tenha um volume de dados que requeira um cluster com mais de 100 nós, recomendamos não usar Cassandra. O time operacional que você precisaria para manter a coisa funcionando apenas não vale a pena. Enquanto produzíamos essa edição do Radar, discutimos várias novas tecnologias de base dados, muitas delas oferecendo avanços de performance dez vezes superiores aos sistemas existentes. Mantemos nosso ceticismo em relação a uma nova tecnologia—principalmente em algo crítico como uma base de dados—até que ela seja propriamente comprovada. Jepsen fornece análise de performance de base de dados sob condições difíceis e encontrou diversos bugs em várias bases de dados NoSQL. Recomendamos manter uma dose saudável de ceticismo e observar de perto sites como Jepsen quando for avaliar a tecnologia de sua base de dados.

À medida que mais organizações escolhem implantar aplicações na nuvem, temos encontrado com regularidade grupos de TI que estão desperdiçando esforços tentando replicar suas abordagens de gerenciamento e segurança de centros de dados na nuvem. Isso frequentemente se apresenta na forma de firewalls, balanceadores de carga, proxies de rede, controle de acesso, aparelhos de segurança e serviços que são estendidos à nuvem sendo minimamente repensados. Temos visto organizações construírem suas próprias APIs de orquestração à frente de seus provedores de nuvem para restringir os serviços que podem ser utilizados por times. Na maioria dos casos, essas camadas servem apenas para prejudicar a capacidade, removendo a maior parte dos benefícios visados ao mover para a nuvem. Nesta edição do Radar, escolhemos destacar novamente a técnica de **subir e migrar para nuvem** como algo a ser evitado. Ao invés disso, as organizações devem olhar com mais cuidado para o objetivo de seus controles existentes de segurança e operação e procurar por controles alternativos que funcionam na nuvem sem criar restrições desnecessárias. Muitos desses controles já existem para provedores de nuvem maduros, e times que adotam a nuvem podem usar APIs nativas para provisionamento e operações self-service.

PLATAFORMAS

Segurança de transporte HTTP estrito **(HSTS)** é atualmente uma política amplamente suportada, que permite que websites se protejam de ataques downgrade. Um ataque downgrade, no contexto do HTTPS, é aquele que pode direcionar usuários do seu site para utilizar HTTP ao invés de HTTPS, permitindo outros ataques, como ataques man-in-the-middle. Com HSTS, o servidor envia um cabeçalho informando ao navegador que ele deve usar apenas HTTPS para acessar o site. O suporte dos navegadores é hoje difundido o suficiente para que essa funcionalidade simples de implementar seja adicionada por qualquer site usando HTTPS. O Observatory do Mozilla pode ajudar a identificar esse e outros cabeçalhos e opções de configuração úteis para melhorar a segurança e a privacidade. Ao implementar HSTS, é fundamental garantir que todos os recursos sejam devidamente carregados através de HTTPS, já que uma vez que a HSTS é ativada, (quase) não há volta até o tempo de expiração. A diretiva para incluir subdomínios deve ser adicionada, mas, mais uma vez, uma verificação minuciosa de que todos os subdomínios suportam transporte seguro é necessária.



A lista de aplicações permitidas provou ser uma das mais eficientes formas de mitigar ataques de intrusão cibernética. Uma maneira conveniente de implementar essa prática altamente recomendada é por meio de **módulos de segurança do Linux**. Com SELinux ou AppArmor incluídos por padrão na maioria das distribuições do Linux, e com ferramentas mais completas como Grsecurity prontamente disponíveis, movemos essa tecnologia para o anel Adote nesta edição. Essas ferramentas ajudam os times a avaliar questões sobre quem tem acesso a quais recursos de hospedagens compartilhadas, incluindo serviços contidos. Essa abordagem conservadora de gerenciamento de acesso vai ajudar os times a construir segurança em seus processos de desenvolvimento de software.

Continuamos a ter experiências positivas implantando a plataforma **Apache Mesos** para gerenciar recursos de cluster para sistemas altamente distribuídos. O Mesos abstrai recursos de computação subjacentes como CPU e armazenamento, visando prover utilização eficiente ao mesmo tempo mantendo isolamento. O Mesos inclui Chronos para execução distribuída e tolerante a falhas de tarefas agendadas, e Marathon para orquestrar processos de longa duração em contêineres.

Temos uma crença crescente de que para a maioria dos cenários raramente vale a pena criar seu próprio

ADOTE
23. Docker
24. HSTS
25. Módulos de segurança do Linux

EXPERIMENTE
26. Apache Mesos
27. Auth0
28. AWS Lambda
29. Kubernetes
30. Pivotal Cloud Foundry
31. Rancher
32. Realm
33. Unity além dos jogos

AVALIE
34. .NET Core
35. Amazon API Gateway
36. Apache Flink
37. AWS Application Load Balancer
38. Cuidado com Cassandra
39. Electron
40. Ethereum
41. HoloLens
42. IndiaStack
43. Nomad
44. Nuance Mix
45. OpenVR
46. Tarantool
47. wit.ai

EVITE
48. CMS como plataforma
49. API Gateways excessivamente ambiciosos
50. Nuvem privada superficial

PLATAFORMAS *continuação*

código de autenticação. O gerenciamento de identidade terceirizado acelera a entrega, reduz erros e tende a possibilitar uma resposta mais rápida a vulnerabilidades recém descobertas. **Auth0** particularmente tem nos impressionado nesse campo por sua facilidade de integração, variedade de protocolos e conectores suportados e API rica de gerenciamento.

Nossos times continuam gostando de usar **AWS Lambda** e estão começando a usar o serviço para experimentar com arquiteturas sem servidor, combinando Lambda com o API Gateway. Recomendamos que as funções Lambda contenham apenas uma quantidade moderada de código. Assegurar a qualidade de uma solução baseada em um emaranhado de muitas funções Lambda é difícil, e tal solução pode não ser um bom custo-benefício. Para necessidades mais complexas, implantações baseadas em contêineres ou VMs ainda são preferíveis. Além disso, encontramos problemas significativos usando Java para funções Lambda, com latências irregulares de vários segundos quando o contêiner Lambda é iniciado. É claro que você pode contornar esse problema usando JavaScript ou Python, e se as funções Lambda não contiverem muito código, a escolha da linguagem de programação não deve importar muito.

Realm é uma base de dados concebida para uso em dispositivos móveis, com seu próprio mecanismo de persistência para alcançar alta performance. Realm é vendido como um substituto para SQLite e Core Data. Vale notar que as migrações não são tão fáceis quanto a documentação do Realm faz parecer. Entretanto, mais e mais times estão escolhendo Realm como o mecanismo de persistência em ambientes de produção para aplicações móveis.

Depois de experimentar anos de crescimento como uma plataforma para desenvolvimento de jogos, **Unity** recentemente se tornou a plataforma de escolha para desenvolvimento de aplicações de Realidade Virtual e Aumentada. Esteja você criando um mundo totalmente imersivo para os dispositivos Oculus ou HTC Vive, uma camada holográfica para sua nova aplicação espacial corporativa ou uma funcionalidade de Realidade Aumentada para seu aplicativo móvel, Unity provavelmente oferece o que você precisa tanto para prototipar quanto para preparar para o lançamento. Muitas pessoas na ThoughtWorks acreditam que Realidade Virtual e Aumentada representam a próxima grande mudança significativa em termos de plataforma de computação e, no momento, Unity é a ferramenta mais importante do conjunto de ferramentas que usamos para desenvolver para essa mudança. Usamos Unity para desenvolver todos os nossos protótipos de Realidade Virtual, bem como funcionalidades de Realidade Aumentada para aplicações para dispositivos específicos e para smartphone/tablet.

.NET Core é um produto modular de código aberto para criação de aplicações que podem ser facilmente implantadas em Windows, macOS e Linux. .NET Core torna possível construir aplicações web multiplataforma usando ASP.NET Core com um conjunto de ferramentas, bibliotecas e frameworks—outra escolha para microsserviços e arquitetura. A comunidade em torno do .NET Core e outros projetos relacionados tem crescido. Novas ferramentas apareceram e evoluíram com rapidez, como Visual Studio. Existem imagens de Docker baseadas tanto em Linux quanto em Windows (Nano Server) com .NET Core que simplificam a aplicação da arquitetura de microsserviços. CoreCLR e CoreFX apareceram em edições anteriores do Radar. Entretanto, há alguns meses a Microsoft anunciou o lançamento do .NET Core 1.0, a primeira versão estável. Vemos boas novas oportunidades, mudanças e uma comunidade vibrante como razões para continuar avaliando esse produto.

Amazon API Gateway é a oferta da Amazon que permite a pessoas desenvolvedoras de software disponibilizar serviços de API para clientes da Internet. Ela oferece as funcionalidades comuns de uma API Gateway como gestão de tráfego, monitoramento, autenticação e autorização. Nossos times têm usado esse serviço como frente para outros recursos da AWS, como AWS Lambda como parte de arquiteturas sem servidor. Continuamos monitorando os desafios apresentados por API Gateways excessivamente ambiciosos, mas nesse momento a oferta da Amazon parece ser leve o suficiente para evitar esses problemas.

A Amazon recentemente lançou o **AWS Application Load Balancer** (ALB), um substituto direto para Elastic Load Balancers apresentado em 2009. ALB suporta inspeção de tráfego Camada 7 e é construído para suportar arquitetura de nuvem moderna. Caso você esteja construindo um sistema baseado em microsserviços usando ECS, os novos balanceadores de carga vão entender de forma direta hospedagem e escala de contêineres, com múltiplos contêineres e portas por estância EC2. O roteamento baseado em conteúdo permite a segmentação de requisições em grupos de servidores alvo, ao mesmo tempo escalando esses

PLATAFORMAS *continuação*

grupos de forma independente. As checagens de saúde realizadas pelos balanceadores de carga estão bastante melhoradas, com a capacidade de capturar métricas detalhadas da performance da aplicação. Gostamos de tudo o que vemos aqui, e os times já começaram a reportar uso bem sucedido do ALB.

A base de dados Cassandra da Apache é uma solução de Big Data poderosa e escalável para armazenamento e processamento de grandes quantidades de dados, com frequência usando centenas de nós distribuídos entre múltiplas localidades pelo mundo. É uma ótima ferramenta e nós gostamos dela, mas muito frequentemente vemos times encontrarem problemas usando ela. Recomendamos **cuidado com Cassandra**. Os times frequentemente entendem mal o caso de uso para Cassandra, tentando usá-la como um banco de dados de propósito geral, quando na verdade ele é otimizado para leituras mais rápidas em grandes conjuntos de dados baseada em chaves ou índices pré-definidos. Sua dependência no esquema de armazenamento também pode dificultar a evolução ao longo do tempo. Cassandra também possui uma complexidade operacional significativa, então, a menos que você absolutamente precise da capacidade de escalar que ela oferece, uma solução mais simples normalmente é melhor. Se você não precisa do caso de uso e características de escala específicos do Cassandra, pode ser que você esteja o escolhendo por inveja de Big Data. O uso cuidadoso de Cassandra inclui testes extensivos automatizados, e ficamos felizes em recomendar CassandraUnit como parte de sua estretégia de testes.

Electron é um framework sólido para construção de clientes desktop nativos usando tecnologias web como HTML, CSS e JavaScript. Os times podem alavancar seu conhecimento web para entregar clientes desktop multiplataforma bem acabados sem gastar tempo aprendendo outro conjunto de tecnologias.

A excitação com blockchain e criptomoedas parece ter atingido seu pico, já que os anúncios de grande impacto nessa área têm sido cada vez mais raros, e é esperado que alguns dos esforços mais especulativos se extingam com o tempo. Um dos blockchains, **Ethereum** tem conseguido bom progresso e vale a pena acompanhar. Ethereum é um blockchain público com uma linguagem de programação embutida que permite que "contratos inteligentes" sejam criados. Estes são movimentos algorítmicos do "ether" (criptomoeda do Ethereum) em resposta à atividade ocorrendo no blockchain. R3Cev, o consórcio de tecnologia de blockchain para bancos, construiu suas primeiras provas de conceito no Ethereum. O Ethereum foi usado para construir uma Organização Autônoma Distribuída (Distributed Autonomous Organization ou DAO)—uma das primeiras "corporações algorítmicas"—, embora um assalto recente de 150 milhões de dólares em Ether demonstre que blockchain e criptomoedas ainda são o Velho Oeste do mundo da tecnologia.

Com **HoloLens**, a Microsoft entregou o primeiro dispositivo de realidade aumentada (RA) verdadeiramente utilizável. Não apenas é uma linda peça de design industrial e um dispositivo extremamente confortável de se usar, como também demonstra claramente a promessa de RA para empresas por meio de sua ótica deslumbrante e integração profunda com Windows 10. Esperamos que HoloLens seja a primeira plataforma de RA na qual entreguemos funcionalidades substanciais de aplicação para nossos clientes a curto prazo, e ansiamos por sua evolução à medida que ela ganhar maior tração.

IndiaStack é um conjunto de APIs abertas concebidas com o objetivo de transformar a Índia, de um país pobre em dados para um país rico em dados. A pilha de tecnologia enfatiza inovação em camadas, especificando um conjunto mínimo de APIs e encorajando o restante do ecossistema a construir aplicações personalizadas com base nessas APIs. Aadhaar funciona como uma das camadas de base, fornecendo serviços de autenticação para mais de um bilhão de cidadãos indianos. Além disso, existem serviços para fornecer transações digitais por meio de assinaturas digitais (eSign), pagamento online unificado (UPI) e uma camada de consentimento eletrônico (e-KYC) para fornecer de forma segura detalhes do Aadhaar para provedores de serviço. Acreditamos na iniciativa guiada por API abertas para trazer inovação digital, e os princípios de design por trás do IndiaStack podem ser usados como agentes de mudança para outras regiões e países.

Nuance Mix é um framework para processamento de linguagem natural da mesma empresa que criou a tecnologia de conversão de voz em texto por trás do Dragon Speaking e da primeira versão do Siri. O framework suporta a criação de gramáticas que permitem qualquer tipo de interação com usuários por meio de voz. A pessoa que estiver desenvolvendo define uma gramática específica por domínio a qual o framework pode se treinar para entender. Os resultados são respostas às entradas do usuário que identificam as intenções do usuário e conceitos de interação. Em

PLATAFORMAS *continuação*

um primeiro momento, ele se limita a frases próximas às usadas para treiná-lo, mas com o tempo ele pode começar a identificar o significado de frases diferentes. Embora ainda esteja em versão beta, a acurácia nas primeiras explorações tem sido convincente, e o produto final é uma opção a se observar quanto a formulários de aplicação que podem se beneficiar da interação com o usuário sem usar as mãos—incluindo os campos móvel, IoT, RA, RV e interativo.

OpenVR é o SDK subjacente para fazer muitos dos head-mounted displays (HMDs) de RV funcionarem com Unity e provavelmente vai continuar a crescer em importância. Boa parte do trabalho de RV na ThoughtWorks foi construído com base no OpenVR, porque ele pode ser executado em qualquer HDM, ao contrário de outros SDKs. Embora não tenha código aberto, é gratuito por meio de sua licença. O SDK do Oculus é mais restritivo em seu processo de licenciamento e só funciona em dispositivos Oculus. OSVR, embora tenha verdadeiramente código aberto, não parece ter grande adoção ainda. Se você for desenvolver uma aplicação de RV para o maior número de dispositivos possível—e sem usar Unity ou Unreal para desenvolvê-la—OpenVR é a solução mais concreta e pragmática no momento.

Tarantool é uma solução NoSQL de código aberto que combina base de dados e cache e fornece APIs para escrever lógica de aplicação em Lua. Tanto mecanismos em memória quanto baseados em disco são suportados, e os usuários podem criar múltiplos índices (HASH, TREE, RTREE, BITSET) baseados em seus casos de uso. Os dados em si são armazenados em formato MessagePack e usam o mesmo protocolo para comunicação entre clientes e servidor. Tarantool suporta logs write-ahead, transações e replicação master-master assíncrona. Estamos felizes com a decisão arquitetural de adotar a política single-writer e multitarefa cooperativa para lidar com conexões simultâneas.

A excitação em torno da inteligência de máquina atingiu um pico, mas, assim como no caso de Big Data, frameworks e ferramentas úteis estão esperando ser descobertos no meio disso tudo. Uma dessas ferramentas é **wit.ai**, uma plataforma SaaS que permite a quem desenvolve criar interfaces de conversação usando processamento de linguagem natural (PLN). Wit funciona com entradas tanto de texto quanto de voz, ajuda quem desenvolve a gerenciar intenção de conversação e permite que lógica de negócios personalizada seja implementada usando JavaScript. O sistema é gratuito para uso comercial e não-comercial e incentiva a criação de aplicações abertas. Esteja ciente de que você precisa aceitar que o Wit use seus dados para melhorar o serviço e para análise própria, então leia os termos e condições com cuidado. Outro candidato nesse campo é o Microsoft Bot Framework, mas ele está disponível apenas em forma de preview limitada até o momento dessa publicação. Assim como a maioria das coisas da Microsoft, esperamos que o Bot Framework evolua com rapidez, então vale a pena observar de perto.

Temos visto muitas organizações encontrarem problemas na tentativa de usar seu **CMS como plataforma** para entrega de aplicações digitais grandes e complexas. Isso é com frequência motivado pela esperança alimentada por fornecedores de ignorar as organizações de TI que não respondem e permitir que o negócio arraste e solte alterações diretamente em produção. Ainda que apoiemos firmemente o fornecimento de ferramentas e fluxos de trabalho certos para produtores de conteúdo, para aplicações com lógica de negócio complexa tendemos a recomendar tratar seu CMS como um componente da sua plataforma (muitas vezes em modo híbrido ou headless) cooperando de forma limpa com outros serviços, ao invés de tentar implementar toda a sua funcionalidade no CMS em si.

Uma das nossas reclamações comuns é sobre colocar inteligência de negócio em middleware, o que resulta software de transporte que ambicionam executar uma lógica de aplicação crítica. Fornecedores no competitivo mercado de API Gateways continuam adicionando funcionalidades que diferenciam seus produtos. Isso tem como resultado produtos com **API Gateways excessivamente ambiciosos**, cuja funcionalidade—baseada essencialmente em um proxy reverso—incentiva designs que são difíceis de testar e implantar. API Gateways podem ser úteis para lidar com algumas preocupações genéricas – por exemplo, autenticação e limitação de carga – mas quaisquer inteligências de domínio como transformação de dados ou processamento de regras devem residir em aplicações ou serviços nos quais eles possam ser controlados por times de produto trabalhando próximos aos domínios que eles suportam.

FERRAMENTAS

Babel.js se tornou o compilador padrão para escrever JavaScript da próxima geração. Seu ecossistema está realmente decolando, graças a seu sistema de plugins reestruturado. Ele permite a quem desenvolve escrever código ES6 (e até mesmo ES7) que é executado no navegador e no servidor sem sacrificar a compatibilidade com versões anteriores para navegadores antigos, e com pouquíssima configuração. Possui suporte de primeira classe para diferentes sistemas construa-e-teste, o que torna simples a integração com qualquer fluxo de trabalho em uso. É um ótimo software que se tornou o principal motivador da adoção e da inovação do ES6 (e do ES7).

Quando combinam técnicas e estilos de arquitetura modernos, como microsserviços, DevOps e QA em produção, times de desenvolvimento precisam de monitoramento cada vez mais sofistifcado. Simplesmente olhar para gráficos de uso do disco e de utilização da CPU não é mais suficiente, e muitos times coletam métricas da aplicação e específicas do negócio usando ferramentas como Graphite e Kibana. **Grafana** torna mais fácil a criação de paineis úteis e elegantes de dados extraídos de diversas fontes. Uma funcionalidade particularmente útil permite a sincronização de calendários de diferentes gráficos, o que ajuda a identificar correlações nos dados subjacentes. O sistema de templates que está sendo adicionado promete bastante, e provavelmente vai tornar o gerenciamento de conjuntos de serviços similares ainda mais fácil. Devido às suas forças, Grafana se tornou nossa escolha padrão nessa categoria.

Imagens de máquina se tornaram fundamentais em pipelines de implantação modernas, e há uma diversidade de ferramentas e técnicas para criar as imagens. Devido a seu conjunto completo de funcionalidades e às experiências positivas que tivemos com ele, recomendamos **Packer** mais do que outras alternativas. Também recomendamos não tentar escrever scripts personalizados para fazer o que Packer faz.

No topo da pirâmide de testes para desenvolvimento de aplicações Android, nossos times estão cada vez mais usando **Espresso** como a ferramenta de testes funcionais. Sua API principal sucinta esconde os detalhes de implementação confusos e ajuda a escrever testes concisos, com execução mais rápida e confiável. Com Espresso, você pode executar testes de UI automatizados simulando interações de usuários em um único aplicativo de destino, tanto em emuladores quanto em dispositivos reais com diferentes versões do Android.

ADOTE
51. Babel
52. Consul
53. Grafana
54. Packer

EXPERIMENTE
55. Apache Kafka
56. Espresso
57. fastlane
58. Galen
59. HashiCorp Vault
60. JSONassert
61. Let's Encrypt
62. Load Impact
63. OWASP Dependency-Check
64. Pa11y
65. Serverspec
66. Talisman
67. Terraform
68. tmate
69. Webpack
70. Zipkin

AVALIE
71. Android-x86
72. axios
73. Bottled Water
74. Clojure.spec
75. FBSnapshotTestcase
76. Grasp
77. LambdaCD
78. Pinpoint
79. Pitest
80. Repsheet
81. Scikit-learn

EVITE
82. Jenkins como uma pipeline de implantação

FERRAMENTAS *continuação*

fastlane é nossa ferramenta favorita para automatizar a maioria das atividades entediantes envolvidas na criação, testes, documentação e provisionamento de aplicativos móveis para iOS e Android. Configuração simples, uma variedade de ferramentas e múltiplos pipelines fazem dela um ingrediente chave para entrega contínua para aplicações móveis.

Testar se layout e estilo de websites responsivos estão funcionando como esperado em vários dispositivos diferentes pode ser um processo lento e frequentemente manual. **Galen** ajuda a amenizar esse problema oferecendo uma linguagem simples, executada em Selenium, que permite que você especifique expectativas para a aparência do seu website em vários tamanhos de tela. Embora Galen sofra com problemas de fragilidade e velocidade típicos de qualquer abordagem de teste de ponta a ponta, encontramos benefícios em ter rápido feedback sobre questões de design.

Ter uma forma de armazenar segredos com segurança tem se tornado uma questão cada vez mais importante para projetos. A velha prática de manter segredos em um arquivo ou variáveis de ambiente está se tornando difícil de se administrar, principalmente em ambientes com múltiplas aplicações e grande número de microsserviços. **HashiCorp Vault** lida com o problema fornecendo mecanismos para acessar segredos com segurança através de uma interface unificada. A ferramenta nos atendeu bem em muitos projetos, e nossos times gostaram da facilidade para integrar Vault com seus serviços. Armazenar e atualizar segredos é um pouco difícil, já que depende de uma ferramenta de linha de comando e um bom nível de disciplina do time.

Mais projetos estão emitindo e consumindo informações formadas como JSON. Escrever testes em Java para JSON pode ser trabalhoso. **JSONassert** é uma biblioteca pequena para ajudar a escrever testes menores com JSON simplificando asserções e fornecendo mensagens de erro melhores.

Pa11y é um testador automático de acessibilidade que pode ser executado pela linha de comando e incluído em uma build pipeline. Nossos times obtiveram sucesso usando Pa11y em um site altamente dinâmico criando primeiro uma versão HTML estática, e depois executando os testes de acessibilidade. Para muitos sistemas—principalmente websites governamentais—testes de acessibilidade são uma exigência, e Pa11y torna tudo isso bem mais fácil.

Com a maturidade de ferramentas como Vault, não há mais desculpa para armazenar segredos em repositórios de código, principalmente porque isso acaba sendo frequentemente o ponto fraco de sistemas importantes. Mencionamos anteriormente ferramentas de varredura de repositório como Gitrob, mas atualmente estamos usando ferramentas proativas como **Talisman**, que é um prepush hook para Git que examina commits em busca de segredos correspondentes a padrões predefinidos.

Com **Terraform**, a infraestrutura de nuvem pode ser gerenciada através de definições declarativas. A configuração dos servidores instanciados pelo Terraform geralmente é deixada para ferramentas como Puppet, Chef, ou Ansible. Gostamos do Terraform porque a sintaxe de seus arquivos é bastante legível e porque ele suporta vários provedores de nuvem, ao mesmo tempo em que não tenta criar nenhuma abstração artificial. Dando seguimento a nossa primeira e mais cautelosa menção ao Terraform, há quase dois anos, a ferramenta passou por um desenvolvimento contínuo e evoluiu para um produto estável que provou seu valor em nossos projetos. O problema com gerenciamento de estados pode agora ser evitado usando o que o Terraform chama de "backend de estado remoto". Usamos Consul com sucesso para esse propósito.

A programação em pares é uma técnica essencial para nós, e—dado que estamos vendo mais e mais times com membros distribuídos em diversos lugares— temos experimentado várias ferramentas para dar suporte ao pareamento remoto. Certamente gostamos do ScreenHero, mas temos preocupações quanto a seu futuro. Para times que não dependem de um IDE gráfico, usar **tmate** para pareamento acabou se tornando uma ótima solução. tmate é um fork da popular ferramenta tmux, e comparado a tmux para pareamento remoto, a configuração é muito mais fácil. Comparados a soluções gráficas de compartilhamento de tela, os requisitos de largura de banda e recursos são modestos, e obviamente ele nunca apresenta tela desfocada. Os times também podem configurar seu próprio servidor, mantendo assim o controle total da privacidade e a integridade da solução.

FERRAMENTAS *continuação*

Android-x86 é uma adaptação do projeto código aberto Android para plataformas x86. O projeto começou hospedando vários patches da comunidade para suporte para x86, mas em seguida criou sua própria base de código para oferecer suporte para diferentes plataformas x86. Observamos economia de tempo significativa com o uso de Android-x86 em nossos servidores de integração contínua no lugar de emuladores para testes herméticos de interface do usuário. Entretanto, para testes específicos de interface do usuário direcionados a uma resolução de dispositivo específica—simulando baixa quantidade de memória, largura de banda e bateria—é melhor ficar com os emuladores.

Nossos times têm tido sucesso com **axios**, um cliente HTTP baseado em promessas em JavaScript que eles descrevem como "melhor que Fetch". O projeto tem muitos endossos e atividade no GitHub, e tem nossa aprovação.

Com o aumento do interesse em arquiteturas de fluxo contínuo de dados e os lagos de dados derivados que elas alimentam, temos visto um aumento de confiança em ferramentas que alteram a captura de dados para conectar bancos de dados transicionais a sistemas de processamento de fluxo. **Bottled Water** é uma adição bem-vinda a esse campo, convertendo alterações no log write-ahead do PostgreSQL em eventos Kafka. Uma desvantagem dessa abordagem, entretanto, é que você se prende a eventos do banco de dados de baixo nível ao invés de eventos de negócios de maior nível, que recomendamos como base para uma arquitetura orientada a eventos.

Um dos debates perpétuos envolvendo desenvolvedores envolve a tipagem em linguagens de programação: Exatamente quanto é a medida certa? Clojure, o Lisp funcional dinamicamente tipado na JVM, adicionou uma nova contribuição nessa discussão que torna as coisas mais confusas. **Clojure.spec** é um novo utilitário construído em Clojure que permite a quem desenvolve envolver tipos e outros critérios de verificação em estruturas de dados, por exemplo variação de valores admissíveis. Uma vez definidas, o Clojure usa essas especificações para fornecer uma variedade de benefícios: testes gerados, validação, desestruturação de estruturas de dados, entre outras. Clojure.spec é uma maneira promissora de ter os benefícios da tipagem e das variações onde quem desenvolve precisa deles, mas não em todos os lugares.

Testar a parte visual de aplicações iOS pode ser difícil, lento e incerto, e é por isso que estamos felizes de incluir o **FBSnapshotTestcase** no nosso conjunto de ferramentas. Ele automatiza o processo de tirar, armazenar e diferenciar snapshots de componentes da interface de usuário para que você possa manter suas interfaces perfeitas no nível dos pixels. Como ele executa como um teste de unidade (no simulador), é mais rápido e mais confiável que abordagens de teste funcional.

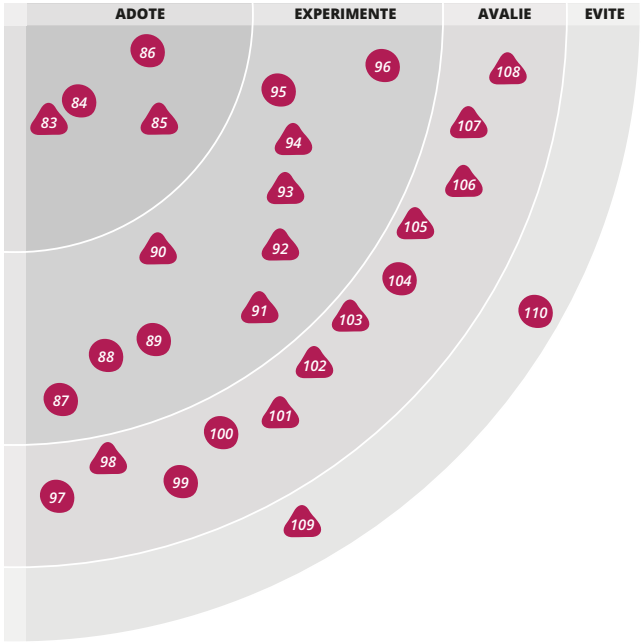
Scikit-learn é uma biblioteca de aprendizagem de máquina escrita em Python cada vez mais popular. Ela fornece um conjunto robusto de modelos de aprendizagem de máquina como clusterização, classificação, regressão e redução de dimensionalidade, e um conjunto rico de funcionalidades para tarefas complementares como seleção de modelos, avaliação de modelos e preparação de dados. Como foi concebida para ser simples, reutilizável em vários contextos e bem documentada, consideramos essa ferramenta acessível até mesmo para não-especialistas explorarem o campo da aprendizagem de máquina.

LINGUAGENS E FRAMEWORKS

Se você está lidando com a construção de uma aplicação de página única e tentando escolher um framework para usar, **Ember.js** surge como a escolha principal. Nossos times elogiam Ember por sua experiência de desenvolvimento altamente produtiva, com muito menos surpresas que outros frameworks como AngularJS. A ferramenta de compilação Ember CLI é um alento no meio do caos das ferramentas de compilação JavaScript, e o time principal e a comunidade Ember são altamente ativos e responsivos.

Com a crescente complexidade de aplicações JavaScript de página única, temos visto uma necessidade maior de deixar previsível o gerenciamento de estado do lado do cliente. **Redux**, com seus três princípios de restrições para atualização de estado, provou ser de grande valor em muitos projetos que implementamos. Os tutoriais primeiros passos com Redux e Redux idiomático são um ponto ponto de partida para usuários novos e experientes. O design minimalista de sua biblioteca gerou um rico conjunto de ferramentas, e encorajamos você a conferir o projeto redux-ecosystem-links para exemplos, middleware e bibliotecas de utilitários. Também gostamos particularmente da testabilidade envolvida: As ações de despacho, as transições de estado e a renderização podem ser testadas unitariamente separadas umas das outras e com quantidades mínimas de mocking.

O interesse na linguagem de programação **Elixir** continua a crescer. Cada vez mais temos visto ela sendo usada em projetos sérios e ouvido feedback de pessoas que desenvolvem que consideram seu modelo de Atores robusto e bastante rápido. Elixir, que é desenvolvida com base na máquina virtual Erlang, tem mostrado potencial na criação de sistemas altamente concorrentes e tolerantes a falhas. Elixir possui funcionalidades características como o operador Pipe, que permite construir uma pipeline de funções como no shell do UNIX. O bytecode compartilhado permite ao Elixir interoperar com Erlang e alavancar bibliotecas existentes, ao mesmo tempo suportando



ferramentas como Mix, para builds, a shell interativa Iex e o framework de testes de unidade ExUnit.

Temos gostado dos testes de interface do usuário rápidos de nível do componente que **Enzyme** oferece para aplicações React.js. Ao contrário de muitos frameworks de teste baseados em snapshot, o Enzyme permite que você teste sem fazer renderização no dispositivo, o que resulta em testes mais rápidos e mais granulares. Esse é um fator que contribui para nossa capacidade de reduzir massivamente a quantidade de testes funcionais que descobrimos que temos que fazer em aplicações React.

A imutabilidade é frequentemente enfatizada no paradigma de programação funcional, e a maioria das linguagens é capaz de criar objetos imutáveis—que não podem ser alterados depois de criados. **Immutable.js** é uma biblioteca para JavaScriptque fornece muitas estruturas

ADOTE
83. Ember.js
84. React.js
85. Redux
86. Spring Boot

EXPERIMENTE
87. Butterknife
88. Dagger
89. Dapper
90. Elixir
91. Enzyme
92. Immutable.js
93. Phoenix
94. Quick e Nimble
95. React Native
96. Robolectric

AVALIE
97. Aurelia
98. ECMAScript 2017
99. Elm
100. GraphQL
101. JuMP
102. Web Física
103. Rapidoid
104. Recharts
105. ReSwift
106. Three.js
107. Vue.js
108. WebRTC

EVITE
109. AngularJS
110. JSPatch

LINGUAGENS E FRAMEWORKS *continuação*

de dados imutáveis persistentes, que são altamente eficientes em máquinas virtuais JavaScript modernas. Os objetos de Immutable. js, entretanto, não são objetos JavaScript normais, portanto, referências a objetos JavaScript a partir de objetos imutáveis devem ser evitadas. Mais times estão usando essa biblioteca para monitorar mutações e manter estados em produção. Recomendamos a quem desenvolve explorar essa biblioteca, principalmente quando combinada com o restante do pacote do Facebook.

Alguns dos times da ThoughtWorks tiveram experiências muito positivas com **Phoenix**, um framework web MVC de lado do servidor escrito em Elixir. Além de ser simples e fácil de usar, o Phoenix se aproveita do Elixir para ser extremamente rápido. Para algumas pessoas, o Phoenix evoca o entusiasmo que elas experimentaram quando descobriram Ruby e Rails. Embora o ecossistema de bibliotecas para Phoenix não seja tão extenso quanto para alguns frameworks mais maduros, ele deve se beneficiar do sucesso e crescimento contínuos do suporte para Elixir.

A maioria dos nossos times de iOS estão usando atualmente o par **Quick** e **Nimble** para seus testes de unidade. Na família RSpec de ferramentas de teste de desenvolvimento guiado por comportamento (BDD), ele fornece testes bastante legíveis (com blocos de descrição) em Swift e Objective-C e tem bom suporte para testes assíncronos.

O **ECMAScript 2017**—não confundir com o ES7 (também conhecido como ECMAScript 2016)—traz diversas melhorias dignas de nota para a linguagem. Espera-se que os navegadores implementem esse padrão integralmente entre junho e agosto de 2017, mas o compilador de JavaScript Babel já suporta várias das funcionalidades atualmente. Se você faz uso extensivo de JavaScript e sua base de código está sob desenvolvimento ativo, recomendamos que você adicione Babel a seu build pipeline e comece a usar as funcionalidades suportadas.

JuMP é uma linguagem específica de domínio para otimizações matemáticas em Julia. JuMP define uma API comum chamada MathProgBase e habilita os usuários a escrever código agnóstico de solvers em Julia. Entre os solvers suportados atualmente estão Artelys, Knitro, Bonmin, Cbc, Clp, Couenne, CPLEX, ECOS, FICO Xpress, GLPK, Gurobi, Ipopt, MOSEK, NLopt e SCS. Um outro benefício é a implementação da técnica de diferenciação automática em modo reverso para computar derivadas de forma que os usuários não ficam limitados a operadores padrão como sin, cos, log e sqrt, mas podem também implementar suas próprias funções objetivo personalizadas em Julia.

Causou a nós curiosidade o padrão **Web Física** criado pelo Google. A ideia da Web Física é simples—beacons publicam uma URL—mas as possibilidades são amplas. Basicamente, essa é uma forma de tomar notas do mundo físico, atrelando objetos e locais ao mundo digital. O mecanismo de transporte atual é Eddystone URLs sobre Bluetooth LE, e clientes de exemplo estão disponíveis. Embora existam preocupações óbvias com segurança ao seguir links descobertos aleatoriamente, temos muito interesse em casos de uso com clientes feitos sob medida nos quais você pode filtrar ou fazer proxy nas URLs como requerido.

Rapidoid é uma coleção de módulos de framework web, incluindo um veloz servidor HTTP de baixo nível implementado do zero com base em Java NIO. Uso inteligente de buffers off-heap de entrada e saída, pools de objetos e estruturas de dados thread-local dão ao Rapidoid uma vantagem sobre outros servidores baseados em NIO, como o Netty. Sendo um projeto relativamente novo, o Rapidoid ainda tem que implementar algumas funcionalidades como cache e suporte SSL. Sugerimos que você cheque o roadmap para atualizações.

Estamos contentes que o paradigma Redux conseguiu traçar seu caminho até a terra do Swift na forma de **ReSwift**. Encontramos benefícios reais na simplicidade e na legibilidade das bases de código uma vez que estado e alterações de estado são gerenciados em um local central e de forma comum. Isso também ajuda na criação de aplicações “offline primeiro”.

Apesar do fervor em torno da onda de novos dispositivos de realidade virtual e aumentada (RV e RA), acreditamos que existem muitos cenários de RV e RA que fazem sentido em navegadores, principalmente em dispositivos móveis. Dada essa tendência, observamos um aumento no uso de **Three.js**, um poderoso framework JavaScript de visualização e renderização 3D. O crescimento do suporte para WebGL, no qual ele é baseado, ajudou na adoção, assim como a vibrante comunidade dando suporte a esse projeto de código aberto.

No mundo de mudanças constantes dos frameworks de front-end JavaScript, **Vue.js** ganhou bastante espaço como alternativa leve ao AngularJS. Ele foi concebido para ser uma biblioteca bastante flexível—e menos opinativa—que oferece um conjunto de ferramentas para construção de interfaces web interativas em torno de conceitos como modularidade, componentes e fluxos de dados reativos. Ele possui um nível baixo de dificuldade de aprendizagem, o que o torna interessante para quem está começando a desenvolver. O Vue.js em si não é um framework maduro. É focado apenas na camada de visualização e, portanto, fácil de integrar com outras bibliotecas ou projetos existentes.

A adoção generalizada de RA e RV como meios de colaboração e comunicação requer uma moderna e prontamente disponível plataforma de transmissão de vídeo. **WebRTC** é um padrão emergente para comunicação em tempo real entre navegadores que permite transmissão de vídeo por meio de tecnologias web geralmente disponíveis. A variedade de navegadores que suportam esse padrão está crescendo, mas a Microsoft e a Apple tem sido lentas na adoção de WebRTC em seus navegadores próprios. Se o ímpeto continuar a crescer, o WebRTC pode constituir a futura base para a colaboração em RA e RV na web.

O **AngularJS** ajudou a revolucionar o mundo das aplicações JavaScript de página única, e entregamos muitos projetos com sucesso com ele ao longo dos anos. Entretanto, não estamos mais o recomendando (v1) para times começando novos projetos. Preferimos a velocidade de aceleração e as bases de código mais sustentáveis que temos visto com Ember e React, principalmente em conjunto com Redux.

A ThoughtWorks é uma empresa de software e uma comunidade de pessoas apaixonadas e guiadas por propósitos, especialistas em consultoria, entrega e produtos de software. Pensamos de forma disruptiva para entregar tecnologia que atenda aos maiores desafios de clientes, ao mesmo tempo que buscamos revolucionar a indústria de TI e promover mudanças sociais positivas. Criamos ferramentas pioneiras para times de desenvolvimento que aspiram a ser grandiosos.

Nossos produtos ajudam organizações a melhorar constantemente e a entregar software de qualidade para suas necessidades mais críticas. Fundada há mais de 20 anos, a ThoughtWorks cresceu de um pequeno grupo em Chicago para uma empresa de mais de 4000 pessoas, espalhadas em 40 escritórios e em 14 países: África do Sul, Alemanha, Austrália, Brasil, Canadá, China, Equador, Espanha, Estados Unidos, Índia, Itália, Reino Unido, Singapura e Turquia.

ThoughtWorks®