JANEIRO 2014

TECHNOLOGY RADAR

Elaborado pelo Conselho Consultivo de Tecnologia da ThoughtWorks

thoughtworks.com/radar

ThoughtWorks®

O QUE HÁ DE NOVO?

Aqui estão as tendências em destaque nesta edição:

- **Alerta e recuperação antecipados em produção** - Estamos observando uma infinidade de novas ferramentas e técnicas para capturar logs, monitorar, armazenar e consultar dados operacionais. Quando combinadas com uma rápida recuperação oferecida pela virtualização e automação de infraestrutura, as empresas podem reduzir a quantidade de testes necessários antes da implantação, talvez até mesmo executando os testes no próprio ambiente de produção.

- **Privacidade versus Big Data** - Ao mesmo tempo em que estamos animados com as novas perspectivas de negócio possibilitadas pela coleta exaustiva de dados, também estamos preocupados com o fato de muitas empresas armazenarem grande quantidade de dados pessoais desnecessariamente. Defendemos que as empresas adotem uma atitude de "datensparsamkeit" e armazenem apenas o mínimo de informações pessoais necessárias sobre seus clientes.

- **O rolo compressor do JavaScript continua** - O ecossistema em torno do JavaScript continua evoluindo como uma importante plataforma de aplicação. Muitas ferramentas interessantes têm surgido para testes, gerenciamento de dependências e construção de aplicações JavaScript, tanto do lado servidor, como do cliente.

- **A fusão do mundo físico e digital** - dispositivos de baixo custo, plataformas de hardware aberto e novos protocolos de comunicação estão levando a experiência de usar um computador para mais perto do mundo ao nosso redor, longe das telas. Um grande exemplo é a proliferação de dispositivos vestíveis para rastrear dados biométricos e o suporte de hardware em outros dispositivos móveis, como telefones e tablets, para interagir com os mesmos.

'ThoughtWorkers' são apaixonados por tecnologia. Nós desenvolvemos, pesquisamos, testamos, contribuímos com código livre, escrevemos, e visamos a sua constante melhoria - para todos. A nossa missão é liderar e promover a excelência de software e revolucionar a indústria de TI. Nós criamos e compartilhamos o Radar de Tecnologias 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, criou o radar. Eles se reúnem 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 à uma ampla gama de interessados, de CIOs até desenvolvedores. O conteúdo é concebido para ser um resumo conciso. Nós o encorajamos a explorar essas tecnologias para obter mais detalhes. O radar é gráfico por natureza, agrupando os itens em técnicas, ferramentas, plataformas, e linguagens e frameworks. Quando itens do radar podem ser classificados em mais de um quadrante, escolhemos aquele que parece mais adequado. Além disso, nós agrupamos esses itens em quatro anéis para refletir a nossa posição atual dentro deles:

- **Adote:** Acreditamos firmemente que a indústria deveria adotar esses itens. Nós os usamos quando são apropriados em nossos projetos.
- **Experimente:** 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.
- **Avalie:** Vale a pena explorar com o objetivo de compreender como isso afetará sua empresa.
- **Evite:** 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 que os itens têm tomado. Nos interessamos em muito mais itens do que seria razoável 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 http://martinfowler.com/articles/radar-faq.html

O RADAR

TÉCNICAS

***ADOPT* - ADOTE**
1 Capturing client-side JavaScript errors
2 Continuous delivery for mobile devices
3 Mobile testing on mobile networks
4 Segregated DOM plus node for JS Testing
5 Windows infrastructure automation

***TRIAL* - EXPERIMENTE**
6 Capture domain events explicitily
7 Client and server rendering with same code
8 HTML5 storage instead of cookies
9 Instrument all the things
10 Masterless Chef/Puppet
11 Micro-services
12 Perimeterless enterprise
13 Provisioning testing
14 Structured Logging

***ASSESS* - AVALIE**
15 Bridging physical and digital worlds with simple hardware
16 Collaborative analytics and data science
17 Datensparsamkeit
18 Development environments in the cloud
19 Focus on mean time to recovery
20 Machine image as a build artifact
21 Tangible interaction

***HOLD* - EVITE**
22 Cloud lift and shift
23 Ignoring OWASP Top 10
24 Siloed metrics
25 Velocity as productivity

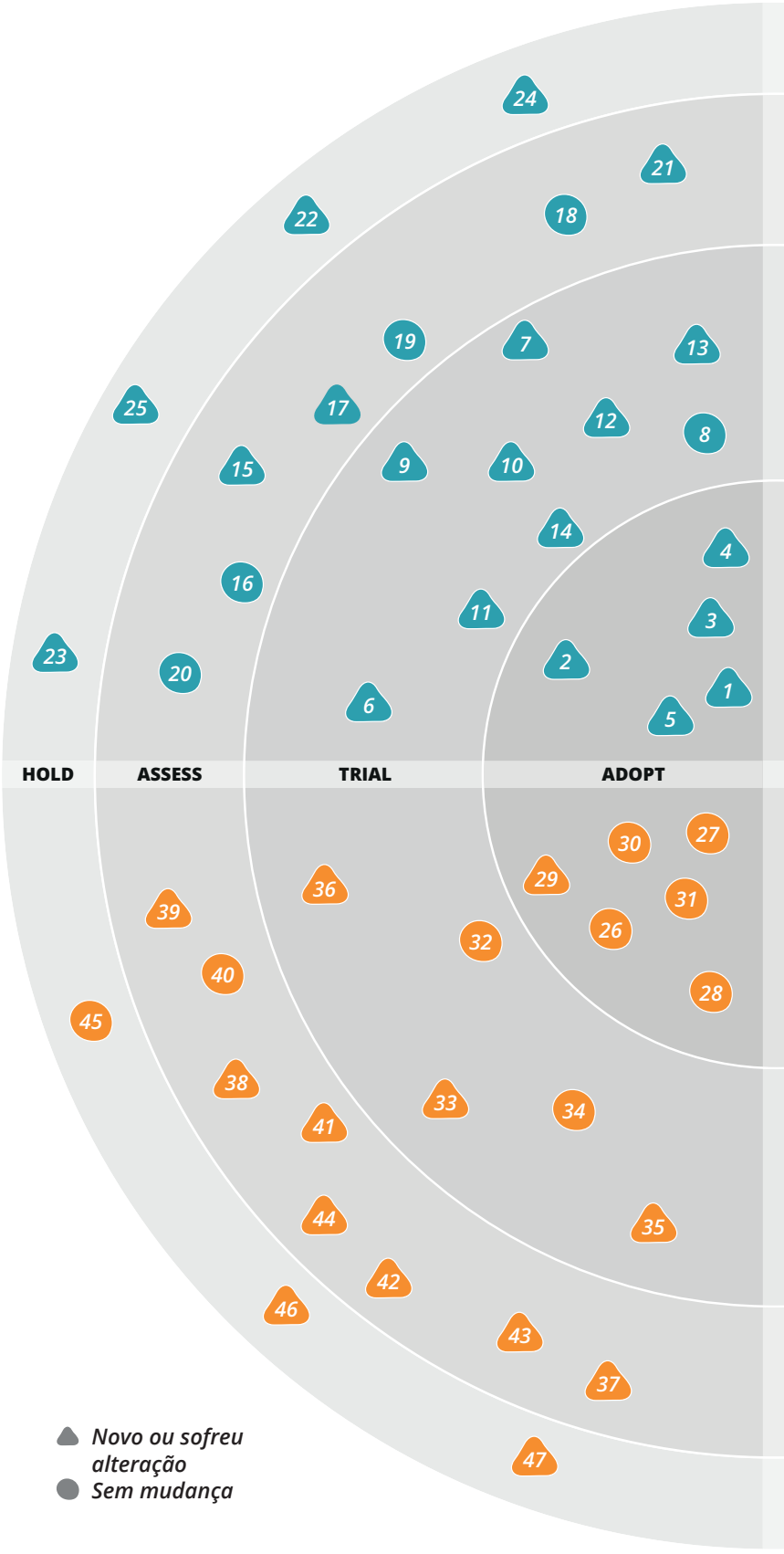
PLATAFORMAS

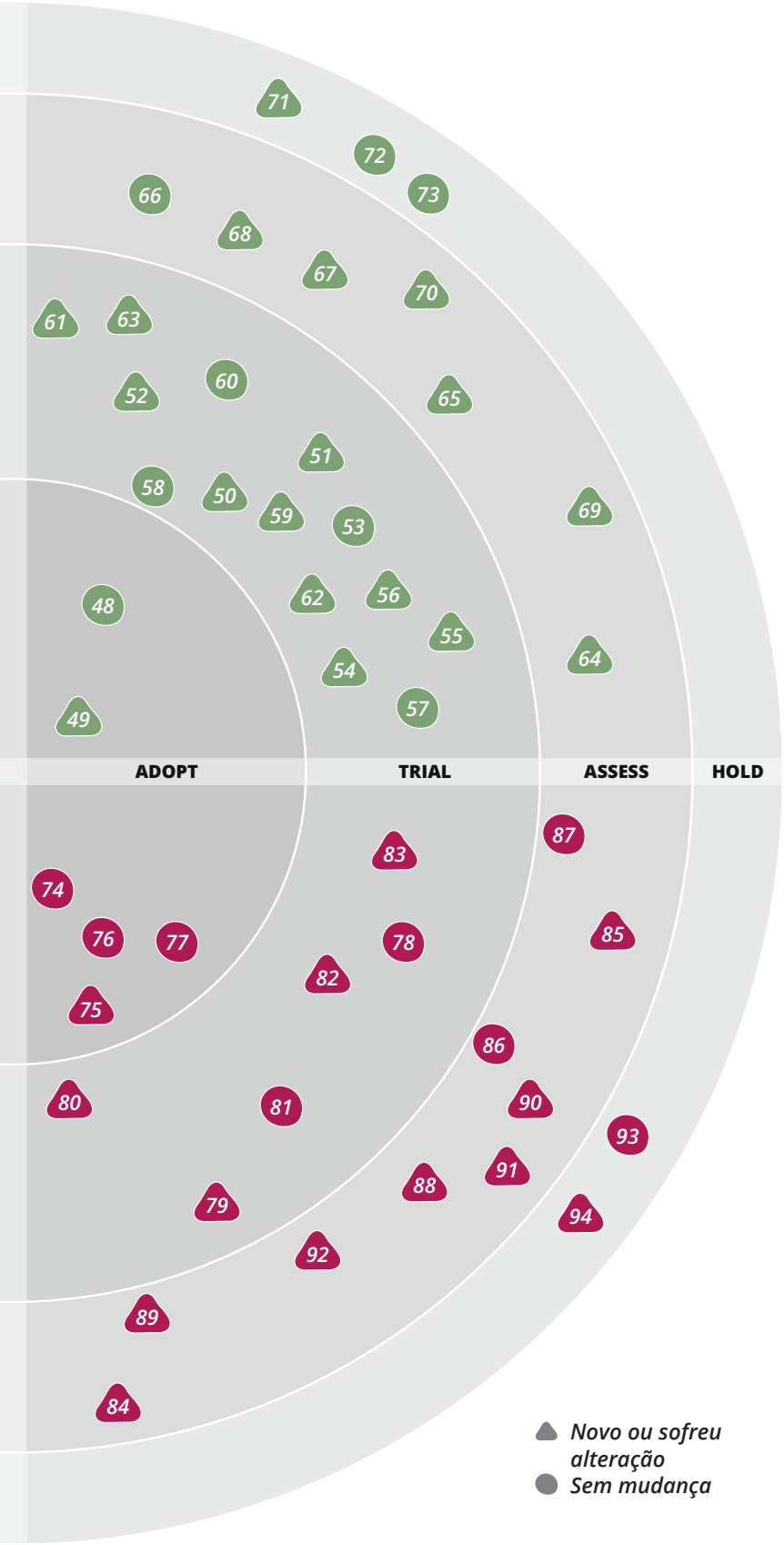
***ADOPT* - ADOTE**
26 Elastic Search
27 MongoDB
28 Neo4J
29 Node.js
30 Redis
31 SMS and USSD as a UI

***TRIAL* - EXPERIMENTE**
32 Hadoop 2.0
33 Hadoop as a service
34 OpenStack
35 PostgreSQL for NoSQL
36 Vumi

***ASSESS* - AVALIE**
37 Akka
38 Backend as a service
39 Low-cost robotics
40 PhoneGap/Apache Cordova
41 Private Clouds
42 SPDY
43 Storm
44 Web Components standard

***HOLD* - EVITE**
45 Big enterprise solutions
46 CMS as a platform
47 Enterprise Data Warehouse





O RADAR

FERRAMENTAS

***ADOPT* - ADOTE**
48 D3
49 Dependency management for JavaScript

***TRIAL* - EXPERIMENTE**
50 Ansible
51 Calabash
52 Chaos Monkey
53 Gatling
54 Grunt.js
55 Hystrix
56 Icon fonts
57 Librarian-puppet and Librarian-Chef
58 Logstash & Graylog2
59 Moco
60 PhantomJS
61 Prototype On Paper
62 SnapCI
63 Snowplow Analytics & Piwik

***ASSESS* - AVALIE**
64 Cloud-init
65 Docker
66 Octopus
67 Sensu
68 Travis for OSX/iOS
69 Visual regression testing tools
70 Xamarin

***HOLD* - EVITE**
71 Ant
72 Heavyweight test tools
73 TFS

LINGUAGEM & FRAMEWORKS

***ADOPT* - ADOTE**
74 Clojure
75 Dropwizard
76 Scala, the good parts
77 Sinatra

***TRIAL* - EXPERIMENTE**
78 CoffeeScript
79 Go language
80 Hive
81 Play Framework 2
82 Reactive Extensions across languages
83 Web API

***ASSESS* - AVALIE**
84 Elixir
85 Julia
86 Nancy
87 OWIN
88 Pester
89 Pointer Events
90 Python 3
91 TypeScript
92 Yeoman

***HOLD* - EVITE**
93 Handwritten CSS
94 JSF

TÉCNICAS

Capturar erros de JavaScript do lado do cliente tem ajudado nossos times de desenvolvimento a identificar problemas específicos de certo navegador ou configurações de plug-in que impactam a experiência do usuário. Durante o ano passado, diversos serviços começaram a suportar esse requisito. Ao invés de armazenar esses erros na base de dados da aplicação, é possível registrar esses dados diretamente em ferramentas de análise e monitoramento existentes, tais como o New Relic.

Desde o último radar, alguns avanços têm tornado menos trabalhoso implementar **a entrega contínua** de aplicativos nativos para **dispositivos móveis**. O Xctool, o "xcodebuild melhor", que recentemente teve seu código aberto, melhora a automação de build iOS e testes de unidade. A chegada de atualizações automáticas no iOS7 reduz o atrito de entregas frequentes. O Travis-CI agora suporta agentes OS X, removendo outro obstáculo para implementar pipelines de entrega contínua para plataformas móveis. Nosso conselho do último radar, sobre o valor de abordagens híbridas e a importância da automação de testes para dispositivos móveis, ainda se aplica.

À medida que aplicações JavaScript do lado do cliente ficam mais sofisticadas, aumenta a necessidade de sofisticação de engenharia para construí-las. Uma falha de arquitetura comum é o acesso irrestrito ao DOM por toda a base de código, misturando a manipulação do DOM com a lógica da aplicação e chamadas AJAX. Isso torna o código difícil de entender e estender. Um bom antídoto é pensar na separação de interesses. Trata-se de restringir de forma agressiva todo o acesso DOM (que geralmente significa qualquer uso do jQuery) para uma fina "camada de segregação". Um efeito colateral positivo dessa abordagem é que tudo fora da **camada segregada DOM** pode ser facilmente testado de forma isolada do navegador.

Utilizar técnicas como "instrumentar todas as coisas" e capturar logs de forma semântica pode ser muito útil para **capturar eventos de domínio explicitamente**. Você pode evitar a necessidade de inferir a intenção do usuário através de transições de estado, modelando essas transições como interesses de primeira classe. Uma forma de alcançar isto é utilizando uma arquitetura baseada em *Event Sourcing*, com os eventos da aplicação sendo mapeados para eventos de negócio significativos.

Cada vez mais o HTML é renderizado não só no servidor, mas também no lado cliente, no navegador web. Em muitos casos essa divisão de renderização continuará sendo uma necessidade. No entanto, com o aumento da maturidade das bibliotecas de templates em JavaScript, uma abordagem interessante tornou-se viável: **cliente e servidor renderizando com o mesmo código.**

Você não pode reagir a importantes eventos de negócio se eles não forem monitorados. O princípio de **instrumentar todas as coisas** nos encoraja a pensar de forma proativa sobre como atingir isso desde o início do processo de desenvolvimento de software. Com isso, podemos derivar métricas-chave, monitorá-las e relatá-las para melhorar a eficácia operacional.

Os servidores Chef & Puppet são um local central para armazenar as receitas/manifestos que propagam alterações de configuração para as máquinas gerenciadas. Eles também são uma base de dados central para informações sobre os nós e fornecem controle de acesso às receitas/manifestos. A desvantagem de ter esses servidores é que eles se tornam um gargalo quando vários clientes se conectam simultaneamente. Eles são um ponto único de falha e demandam esforço para torná-los robustos e confiáveis. Sendo assim, recomendamos **chef-solo ou puppet standalone** em conjunto com um

***ADOPT* - ADOTE**
1 Capturing client-side JavaScript errors
2 Continuous delivery for mobile devices
3 Mobile testing on mobile networks
4 Segregated DOM plus node for JS Testing
5 Windows infrastructure automation

***TRIAL* - EXPERIMENTE**
6 Capture domain events explicitly
7 Client and server rendering with same code
8 HTML5 storage instead of cookies
9 Instrument all the things
10 Masterless Chef/Puppet
11 Micro-services
12 Perimeterless enterprise
13 Provisioning testing
14 Structured Logging

***ASSESS* - AVALIE**
15 Bridging physical and digital worlds with simple hardware
16 Collaborative analytics and data science
17 Datensparsamkeit
18 Development environments in the cloud
19 Focus on mean time to recovery
20 Machine image as a build artifact
21 Tangible interaction

***HOLD* - EVITE**
22 Cloud lift and shift
23 Ignoring OWASP Top 10
24 Siloed metrics
25 Velocity as productivity

TÉCNICAS *continuação*

sistema de controle de versão, quando o servidor é usado primariamente para armazenar receitas/manifestos. As equipes podem sempre introduzir os servidores em caso de necessidade ou caso percebam que estão reinventando soluções para problemas que os servidores já tenham resolvido.

Cada vez mais estamos ultrapassando os limites da nossa capacidade de adquirir e provisionar hardware. No entanto, com o grande aumento na flexibilidade oferecida, descobrimos que somos limitados pela escala e complexidade do software usado para gerenciar nossos recursos virtuais. O uso de técnicas mais conhecidas no mundo do desenvolvimento de software, tais como TDD, BDD e CI (Integração Contínua), possibilita uma abordagem para gerenciar essa complexidade e nos dá a confiança necessária para fazer alterações em nossa infraestrutura de uma maneira segura, repetível e automatizada. Ferramentas para **Testes de Provisionamento**, como o rspec-puppet, o Test Kitchen e o serverspec, estão disponíveis para a maioria das plataformas.

Tratar *logs* como dados nos proporciona mais conhecimento sobre a atividade operacional dos sistemas que construímos. O uso de **Logs estruturados** (*Structured logging*), que significa usar um formato de mensagem consistente, pré-determinado e contendo informações semânticas, permite que ferramentas como Graylog2 e Splunk produzam insights mais significativos.

A redução de custo, tamanho, consumo de energia e simplicidade dos dispositivos levam o software ao mundo físico. Tais dispositivos muitas vezes contêm pouco mais que um sensor e um componente de comunicação, como Bluetooth de baixa energia ou WiFi. Como engenheiros de software, precisamos expandir nosso pensamento para construir **pontes entre os mundos físico e digital através de hardware simples**. Já estamos vendo isso em carros, casas, no corpo humano, na agricultura, dentre outros. O custo e tempo necessários para a prototipação desses dispositivos está se reduzindo para coincidir com as possíveis iterações rápidas de desenvolvimento de software.

Com nosso desejo de apoiar a constante mudança nos modelos de negócios, de aprender com o comportamento passado e de fornecer a melhor experiência para cada visitante individual, somos tentados a gravar o máximo de dados possível. Ao mesmo tempo, os hackers estão mais ativos do que nunca, com uma quebra de segurança espetacular após a outra, e agora ficamos sabendo da vigilância em massa sem precedentes por parte de agências governamentais. O termo **Datensparsamkeit** vem da legislação de privacidade alemã e descreve a ideia de apenas armazenar informação pessoal que seja absolutamente necessária para o negócio ou para as leis aplicáveis. Alguns exemplos são: em vez de armazenar o endereço IP completo de um cliente em logs, usar apenas os dois ou três primeiros octetos; em vez de registrar acessos com um nome de usuário, usar um token anônimo. Se você nunca armazena as informações, você não precisa se preocupar com a possibilidade de alguém roubá-las.

Com a linha de separação entre hardware e software se tornando mais incerta, vemos cada vez mais a computação tradicional sendo embutida em objetos do cotidiano. Embora os dispositivos conectados estejam agora onipresentes em espaços comerciais, automóveis, casas e locais de trabalho, ainda não entendemos como misturá-los em uma experiência de computação útil, que vá além de uma simples tela. A **interação tangível** é uma disciplina que combina tecnologias de software e hardware, arquitetura, experiência do usuário e design industrial. O objetivo é proporcionar ambientes naturais compostos de objetos físicos onde os seres humanos possam manipular e compreender dados digitais.

Conforme a adoção da nuvem cresce, estamos, infelizmente, observando uma tendência para tratar a nuvem como apenas mais um provedor de hospedagem. Essa tendência de **subir e migrar para nuvem** (*lift and shift*) está sendo incentivada por grandes fornecedores, que estão apenas reposicionando suas atuais ofertas de hospedagem como “nuvem”. Poucos delas oferecem qualquer flexibilidade real ou modelos de cobrança onde você paga somente pelo uso. Se você acha que consegue migrar para a nuvem sem reavaliar sua arquitetura, você provavelmente não está fazendo a coisa certa.

Não se passa uma semana sem que a indústria de TI seja envergonhada por mais uma grande perda de dados, vazamento de senhas, ou violação de um sistema supostamente seguro. Há bons recursos para ajudar a garantir que a segurança seja tratada como um interesse primário durante o desenvolvimento de software, e precisamos parar de ignorá-los. O **OWASP Top 10** é um bom lugar para começar.

TÉCNICAS *continuação*

À medida que mais empresas aumentam sua presença online, observamos uma tendência a capturar **métricas em silos**. Ferramentas são implementadas para reunir e exibir métricas específicas: uma ferramenta para visualização de página e comportamento do navegador, outra para os dados operacionais e outra para consolidar mensagens de *log*. Isso leva a silos de dados e à necessidade de integrar as ferramentas de forma manual para extrair inteligência de negócio, que é crucial para o funcionamento da empresa. Essa é uma divisão do domínio analítico causada pelas ferramentas e que fere a capacidade da equipe em tomar decisões. Uma solução muito melhor é ter uma visão consolidada de análise quase em tempo real, utilizando painéis integrados exibindo informações de domínio relevantes para a equipe.

De todas as abordagens com as quais podemos discordar, equiparar a velocidade com a produtividade tornou-se tão predominante que achamos importante colocá-la no nosso quadrante 'Evite'. Quando usada corretamente, a velocidade permite a incorporação do "tempo de ontem" no processo de planejamento de iteração. A velocidade é simplesmente uma estimativa da capacidade de um determinado grupo em um determinado momento. Ele pode melhorar à medida que uma equipe se ajusta ou através da correção de problemas, como a dívida técnica ou testes não-determinísticos no servidor de *build*. No entanto, como todas as métricas, ela pode ser aplicada de forma abusiva. Por exemplo, os gerentes de projeto excessivamente zelosos insistindo na melhoria contínua de velocidade. Tratar a **velocidade como produtividade** leva a comportamentos contraprodutivos nos times, que priorizarão a otimização da métrica em vez da entrega de software funcionando.

FERRAMENTAS

Usar ferramentas de **Injeção de dependências para JavaScript** tem ajudado nossos times de a lidar com grandes quantidades de código JavaScript, estruturando o código e carregando as dependências em tempo de execução. Embora isso simplifique o esforço na maioria dos casos, o carregamento tardio (*lazy*) dificulta dar suporte ao modo *offline*. Diferentes ferramentas de gerenciamento de dependência têm diferentes vantagens, por isso considere seu contexto para fazer a escolha.

Na categoria de ferramentas para orquestração DevOps, o **Ansible** tem aclamação quase universal no âmbito de projetos da ThoughtWorks. Ele possui ferramentas úteis e abstrações em um bom nível de granularidade.

Em projetos móveis, temos ficado impressionados com a funcionalidade, a evolução gradual das funcionalidades e a maturidade do **Calabash**. É uma ferramenta de automação de testes de aceitação para aplicações Android e iOS, que suporta ferramentas comuns do ecossistema, como o Cucumber. É uma opção atraente para projetos heterogêneos.

Seguindo a nossa recomendação no último radar para considerar um foco na redução do tempo médio de recuperação, queremos ressaltar o **Chaos Monkey** da suíte Simian Army da Netflix. É uma ferramenta que desativa instâncias aleatoriamente no ambiente de produção, quando está em operação normal. Quando executado com monitoramento abrangente e um time em *stand by*, ele ajuda a identificar fraquezas inesperadas no sistema. Isso permite que a equipe de desenvolvimento construa mecanismos de recuperação automáticos previamente, em vez de ter um grande esforço para reagir a interrupções que pegaram todos de surpresa.

Várias equipes da ThoughtWorks desenvolvendo aplicações Node.js estão utilizando o **Grunt** para automatizar a maioria das atividades de desenvolvimento, como minificação, compilação e *linting*. Muitas das tarefas comuns estão disponíveis como plugins no Grunt. Você pode até mesmo gerar a configuração de forma programática, se necessário.

Gerenciar o emaranhado de dependências em um sistema distribuído é complicado e é um problema que mais pessoas estão enfrentando com a mudança para micro-serviços mais

granulares. O **Hystrix** é uma biblioteca para a JVM da Netflix que implementa padrões para lidar com falhas em sistemas dependentes, oferece monitoramento de conexões em tempo real, e mecanismos de *caching* e *batching* para tornar as dependências entre serviços mais eficientes. Em combinação com o hystrix-dashboard e o Turbine, essa ferramenta pode ser usada para construir sistemas mais resilientes e fornecer dados quase em tempo real sobre vazão, latência e tolerância a falhas.

Testar micro-serviços baseados em HTTP pode ser trabalhoso e complicado. Particularmente em dois cenários: no consumo de um grupo de micro-serviços no front-end e na comunicação entre micro-serviços. Para lidar com eles, o **Moco** pode ser útil. Ele é um framework de *stubs* leve para testar endpoints HTTP. Você pode ter um servidor *stub* embutido funcionando com 2 linhas de códigos Java ou Groovy, ou um servidor *stub* independente com algumas linhas de JSON para descrever o comportamento necessário.

Nós temos defendido o uso de protótipos de baixa fidelidade feitos à mão para ilustrar as interações do usuário,

***ADOPT* - ADOTE**

48 D3
49 Dependency management for JavaScript

***TRIAL* - EXPERIMENTE**

50 Ansible
51 Calabash
52 Chaos Monkey
53 Gatling
54 Grunt.js
55 Hystrix
56 Icon fonts
57 Librarian-puppet and Librarian-Chef
58 Logstash & Graylog2
59 Moco
60 PhantomJS
61 Prototype On Paper
62 SnapCI
63 Snowplow Analytics & Piwik

***ASSESS* - AVALIE**

64 Cloud-init
65 Docker
66 Octopus
67 Sensu
68 Travis for OSX/iOS
69 Visual regression testing tools
70 Xamarin

***HOLD* - EVITE**

71 Ant
72 Heavyweight test tools
73 TFS

FERRAMENTAS *continuação*

sem ficarmos presos no âmago da questão do design gráfico. O **Prototype On Paper** é uma ferramenta que permite que protótipos de tela individuais desenhados em papel sejam capturados através da câmera do iOS ou Android e conectados entre si, para permitir testes de interação com o usuário. Isso preenche bem a lacuna entre os protótipos de papel estáticos e de baixa resolução e as técnicas de prototipagem de mais alta fidelidade.

Nós mencionamos o **SnapCI** da ThoughtWorks – um serviço hospedado que fornece pipelines de entrega – na última edição do Radar. Desde então, temos visto muitas equipes usarem com sucesso o SnapCI em seus projetos. Se você precisa de uma solução simples para entrega contínua na nuvem, o SnapCI pode fornecê-la com um clique. Sem hardware, sem problemas.

Com o aumento do controle sobre a privacidade dos dados, mais empresas estão preocupadas com o compartilhamento de dados analíticos com terceiros. O **Snowplow Analytics e o Piwik** são exemplos de plataformas analíticas de código aberto que podem ser hospedadas em servidores próprios e fornecem um conjunto de recursos e *roadmap* promissores.

O **Cloud-init** é uma técnica simples, mas poderosa, para realizar ações em uma instância na nuvem em tempo de inicialização (*boot time*). É particularmente útil quando usada com metadados de instância, para que uma instância recém-criada consiga puxar as configurações, dependências e software necessários para executar um determinado papel. Quando usado em conjunto com o padrão de servidor Imutável ou Phoenix, isso pode criar um mecanismo muito responsivo e leve para o gerenciamento de implementações na nuvem.

O projeto de código aberto **Docker** tem gerado um grande interesse dentro da ThoughtWorks, e está crescendo em força e maturidade. O Docker permite que aplicações sejam empacotadas e publicadas como *containers* portáteis e leves que funcionam de forma idêntica em um laptop ou em um cluster de produção. Ele fornece ferramentas para a criação e gerenciamento de *containers* de aplicação, e um ambiente de *runtime* baseado no LXC (LinuX Containers).

Muitas ferramentas de monitoramento são construídas em torno da ideia de máquina. Nós monitoramos o que a máquina está fazendo e qual o software que está sendo executado nela. Quando se trata de infraestrutura baseada em nuvem, especialmente padrões como Phoenix e servidores Imutáveis, essa é uma abordagem problemática. Máquinas vêm e vão, mas o importante é que os serviços continuem funcionando. O **Sensu** permite que uma máquina registre-se e seja monitorada com base em um determinado papel. Quando não estivermos mais utilizando a máquina, podemos simplesmente cancelar o seu registro.

Todo o desenvolvimento para iOS deve ser realizado no OS X. Devido a restrições técnicas e de licenciamento, rodar diversos servidores OS X não é fácil, nem comum. A despeito dessas dificuldades, o Travis CI, com o apoio do Sauce Labs, oferece agora serviços de integração contínua com base em nuvem para projetos iOS e OS X.

O crescimento da complexidade das aplicações web aumentou a consciência de que, além da funcionalidade, a aparência também deve ser testada. Isso deu origem a uma variedade de **ferramentas visuais de teste de regressão**, incluindo o CSS Critic, o dpxdt, o Huxley, o PhantomCSS e o Wraith. As técnicas variam desde asserções diretas de valores CSS até a comparação real de capturas de tela (*screenshots*). Enquanto esse ainda é um campo em pleno desenvolvimento, acreditamos que testes de regressão visuais devem ser adicionados às pipelines de entrega contínua.

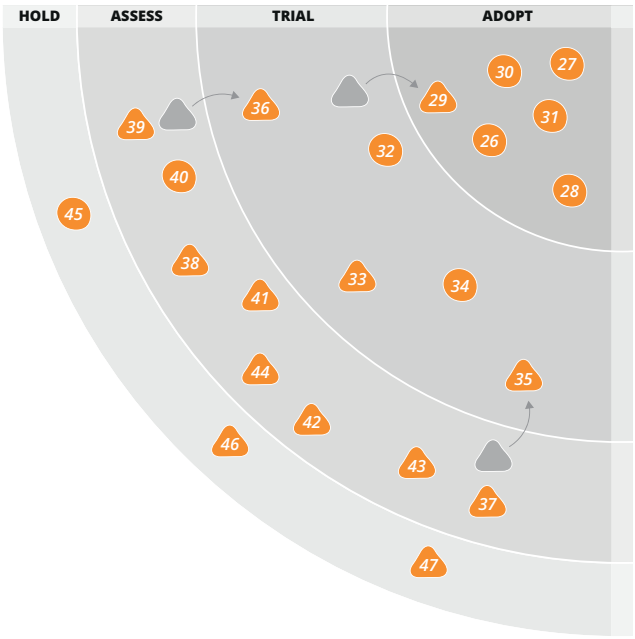
Dentre as várias opções disponíveis para a construção de aplicações móveis multiplataforma, o **Xamarin** oferece um conjunto de ferramentas bastante único. Ele suporta C# e F# como linguagens primárias com chamadas para SDKs de plataformas específicas, além do ambiente de *runtime* Mono, que funciona no iOS, Android e Windows Phone. As aplicações são compiladas para código nativo, em vez da típica abordagem multiplataforma que renderiza uma interface gráfica baseada em HTML em um navegador embutido. Isso dá à aplicação uma aparência e sensação mais nativa. Ao utilizar esse conjunto de ferramentas, a camada de interface do usuário específica da plataforma deve ser separada das outras camadas para garantir o reuso do código entre diferentes plataformas. O binário da aplicação tende a ser um pouco maior, devido ao ambiente de *runtime* que está incluído.

Nós continuamos vendo times despenderem um esforço significativo em scripts de difícil manutenção **Ant** e Nant. Eles são difíceis de entender e estender, pois oferecem pouca expressividade e baixa modularidade. Alternativas como o Gradle, o Buildr e o PSake se demonstraram claramente superiores em termos de manutenção e produtividade.

PLATAFORMAS

Observamos organizações que implantaram o Hadoop com sucesso começando a consolidar seus serviços de infraestrutura Hadoop em uma plataforma centralizada e gerenciada, antes de estender seu uso para o resto da empresa. Essas plataformas **Hadoop como Serviço** (*Hadoop-as-a-Service*) são caracterizadas por uma camada de controle que interage e coordena os diferentes componentes de infraestrutura do núcleo do Hadoop. As capacidades da plataforma são normalmente expostas para o resto da empresa como abstrações de alto nível. Tal plataforma gerenciada permite que as organizações implantem processos, infraestrutura e conjuntos de dados de forma bastante consistente. Esses serviços são construídos em data centers privados e em infraestrutura de nuvem pública.

Akka é um conjunto de ferramentas e runtime para a construção de aplicações altamente concorrentes, distribuídas, orientadas a eventos e tolerantes a falhas em JVM. Ele oferece processos leves baseados em eventos, com aproximadamente 2,7 milhões de atores por GB de RAM e um modelo "*let-it-crash*" de tolerância a falhas projetado para funcionar em um ambiente distribuído. O Akka pode ser usado como uma biblioteca para aplicações web ou como um *kernel* separado, onde a aplicação pode ser inserida.



A recente explosão de produtos focados em dispositivos móveis, além da ampla adoção de abordagens "Lean Start-up" que dão enfase ao tempo de entrega de novas ideias ao mercado, gerou um ecossistema de ofertas de **Backend como serviço** (*Backend-as-a-service ou BaaS*) que permite que os desenvolvedores se concentrem na aplicação cliente e se preocupem menos com interesses do *backend*. Considere adicionar esses serviços ao seu kit de ferramentas quando provar uma ideia de produto rapidamente e com baixo custo for importante. O nosso conselho usual para tomar decisões de desenvolver/comprar/alugar ainda se aplica: tenha clareza sobre quais as áreas funcionais são estratégicas para o seu negócio e quais são secundárias. Para as áreas potencialmente estratégicas, planeje um caminho de migração que permita que você use o provedor BaaS para começar rapidamente, mas que também evite as fricções geradas quando sua arquitetura evoluir e surgir a necessidade de migrá-la para obter essa funcionalidade e personalizá-la como um diferencial.

Com a redução do custo de robôs industriais e o aumento de sua segurança e facilidade de uso, o mundo da robótica comercial está se abrindo. Robôs como o Baxter* da Rethink Robotics, ou o U5 da Universal Robotics, tornam viável para empresas de pequeno a médio porte a automatização de tarefas repetitivas antes desempenhadas por humanos. Cada vez mais o software empresarial terá que se integrar com a **robótica de baixo custo** e considerá-la como mais um participante no fluxo de entrega de valor. O desafio está em tornar a experiência fácil e produtiva para os colegas de trabalho humanos também.

A necessidade de armazenar fisicamente os dados dentro das nações ou organizações tem aumentado significativamente nos últimos anos. Há uma preocupação com a sensibilidade das informações hospedadas em ambientes de nuvem. Organizações estão olhando para a **nuvem privada** como uma alternativa quando os dados precisam ser armazenados em estreita proximidade, com controle de acesso e distribuição. A nuvem privada oferece infraestrutura provisionada,

***ADOPT* - ADOTE**
26 Elastic Search
27 MongoDB
28 Neo4J
29 Node.js
30 Redis
31 SMS and USSD as a UI

***TRIAL* - EXPERIMENTE**
32 Hadoop 2.0
33 Hadoop as a service
34 OpenStack
35 PostgreSQL for NoSQL
36 Vumi

***ASSESS* - AVALIE**
37 Akka
38 Backend as a service
39 Low-cost robotics
40 PhoneGap/Apache Cordova
41 Private Clouds
42 SPDY
43 Storm
44 Web Components standard

***HOLD* - EVITE**
45 Big enterprise solutions
46 CMS as a platform
47 Enterprise Data Warehouse

PLATAFORMAS *continuação*

para uso exclusivo de uma única organização, com as seguintes características: *self-service* sob demanda, amplo acesso à rede, *pool* de recursos, rápida elasticidade e serviço mensurado.

O **SPDY** é um protocolo aberto de rede para o transporte de conteúdo web, com baixa latência, proposto para o HTTP2, que tem visto um aumento no suporte por navegadores modernos. O SPDY reduz o tempo de carregamento da página, priorizando a transferência de sub-recursos de modo que apenas uma conexão por cliente seja necessária. As implementações SPDY usam segurança na camada de transporte (TLS), com transmissão de cabeçalhos gzip ou com compactação *deflate*, em vez de texto legível no HTTP. Ele é ótimo para ambientes com alta latência.

Valores heterogêneos e a grande quantidade de dados não são o único tema do *big data*. Em certas circunstâncias, a velocidade de processamento pode ser tão importante quanto o volume. O **Storm** é um sistema de computação em tempo real distribuído. Ele tem escalabilidade semelhante ao Hadoop, com vazão tão rápida quanto um milhão de tuplas por segundo. Ele permite fazer processamento em tempo real da mesma forma que o Hadoop faz para *batches*.

No radar anterior, nós advertimos contra a utilização de frameworks web tradicionais que fornecem um modelo de componente do lado do servidor. O **padrão de componentes web** (*Web Components*) que se originou no Google, é algo bem diferente. Ele fornece uma maneira mais fácil para criar widgets recicláveis, ajudando a encapsular o HTML, CSS e JavaScript, para que eles não interfiram no resto da página e a página não interfira neles. Os desenvolvedores podem usar muito ou pouco do framework, conforme acharem necessário. Suporte inicial é fornecido pelo projeto Polymer.

Embora a integração centralizada de dados para análise e geração de relatórios continue sendo uma boa estratégia, as iniciativas tradicionais de **Enterprise Data Warehouse** (EDW) têm uma taxa de fracasso maior do que 50%. Gastar muito tempo na modelagem de dados *up-front* resultam em *warehouses* mais complexas que o necessário, que levam anos para serem entregues e com altos custos de manutenção. Nesta edição do radar estamos colocando essas EDWs e técnicas de estilo antigo no anel Evite. Em vez disso, defendemos evoluir seu modelo de dados na direção de um EDW. Teste e aprenda através da construção de pequenos e valiosos incrementos, que são frequentemente colocados em produção. Ferramentas e técnicas não tradicionais podem ajudar, por exemplo, usando um projeto de esquema Data Vault ou mesmo uma base de dados NoSQL como o HDFS.

Sistemas de Gerenciamento de Conteúdo (***Content Management Systems*** ou CMS) têm o seu lugar. Em muitos casos, não faz sentido escrever funcionalidades de edição e fluxo de trabalho a partir do zero. No entanto, nós tivemos sérios problemas quando o uso do **CMS como uma plataforma** (*CMS-as-a-platform*) tornou-se uma solução de TI que evoluiu além do gerenciamento simples de conteúdo.

LINGUAGENS & FRAMEWORKS

Scala é uma grande linguagem que se tornou popular devido à sua facilidade de aproximação por novos desenvolvedores. Essa vasta gama de recursos é um problema, porque muitos aspectos do Scala, como conversões implícitas e dinâmicas, podem trazer complicações. Para usar o Scala com sucesso, você precisa pesquisar a linguagem e ter uma opinião muito forte sobre quais partes são corretas para você, criando a sua própria definição das **melhores partes de Scala** (*Scala, the good parts*). Você pode desativar as partes que não quer, utilizando um sistema chamado marcação de funcionalidades (*feature flags*).

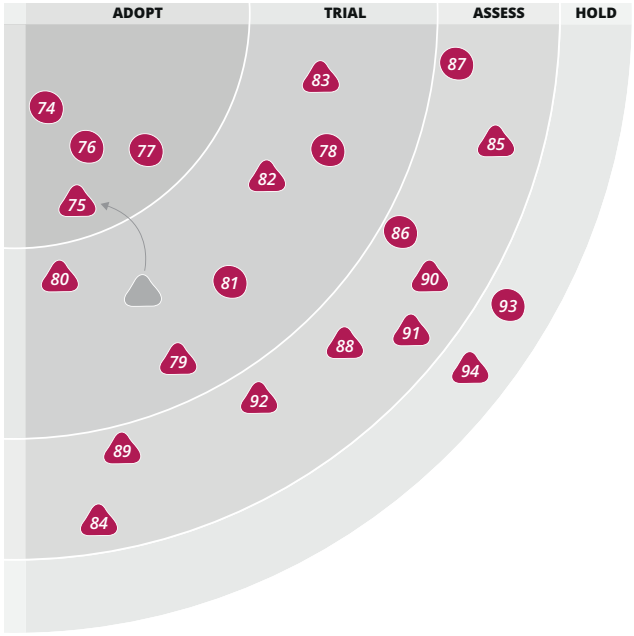
A linguagem **Go** foi originalmente desenvolvida pelo Google como uma linguagem de programação para substituir o C e C++. Quatro anos após o seu lançamento, a linguagem Go está ganhando força em outras áreas. A combinação de binários muito pequenos e estaticamente ligados, com uma excelente biblioteca HTTP, significa que Go tem se tornado popular em organizações que fazem uso de arquiteturas de micro-serviços granulares.

O **Hive** é uma *warehouse* de dados construída em cima do Hadoop que fornece uma linguagem de consulta e definição de dados similar ao SQL, que converte consultas em trabalhos de *map-reduce* executáveis em todo o cluster do Hadoop. Como qualquer abstração útil, o Hive não tenta negar a existência dos mecanismos subjacentes do Hadoop e suporta operações personalizadas de map-reduce como um poderoso mecanismo de extensão. Apesar das semelhanças superficiais com o SQL, o Hive não tenta ser um substituto para mecanismos de consulta de baixa latência em tempo real encontrados em sistemas de banco de dados relacionais. Aconselhamos fortemente a não utilizar o Hive para fins de consultas ad-hoc online.

O blip **Play Framework 2** tem gerado muitas discussões internas. Tivemos sugestões contraditórias para movê-lo tanto para Adote quanto para Evite. Essas diferenças dependem principalmente das aplicações onde ele é usado, como é usado e quais as expectativas que as pessoas têm. Embora nenhuma dessas questões sejam únicas para o Play, ele tem gerado muito mais polêmica do que o esperado no debate "bibliotecas versus frameworks". Reiteramos os cuidados indicados no radar anterior e vamos monitorar como o Play amadurece e dá suporte aos seus pontos fortes.

A Programação Reativa (*Reactive Programming*) lida com streams ou valores que mudam ao longo do tempo. Usando elementos de fluxo de dados, concorrência implícita e propagação transparente de eventos, essas técnicas permitem gerenciar eventos em larga escala de forma eficiente e com baixa latência. No radar anterior, mencionamos as Extensões Reativas (*Reactive Extensions* ou Rx) do .NET devido ao extenso trabalho feito pela Microsoft em fazer do Rx uma parte essencial do framework .NET. Desde então, com a introdução da biblioteca Reactive Cocoa para Objective C; o porte para Java do Reactive Extensions; a biblioteca React para JavaScript; a linguagem Elm baseada em Haskell; e a biblioteca Flapjax para JavaScript, estamos estendendo esse blip para incluir **Extensões Reativas em várias linguagens**.

Até recentemente, o **Web API** era a opção menos pior para construir serviços RESTful utilizando o ASP.NET. O Web API 2 apara diversas arestas com melhor suporte para roteamento flexível, sub-recursos, *media types* e melhor testabilidade. Ele continua a ser a nossa biblioteca preferida para a construção de APIs REST em .NET.



***ADOPT* - ADOTE**
74 Clojure
75 Dropwizard
76 Scala, the good parts
77 Sinatra

***TRIAL* - EXPERIMENTE**
78 CoffeeScript
79 Go language
80 Hive
81 Play Framework 2
82 Reactive Extensions across languages
83 Web API

***ASSESS* - AVALIE**
84 Elixir
85 Julia
86 Nancy
87 OWIN
88 Pester
89 Pointer Events
90 Python 3
91 TypeScript
92 Yeoman

***HOLD* - EVITE**
93 Handwritten CSS
94 JSF

LINGUAGENS & FRAMEWORKS *continuação*

O **Elixir** é uma linguagem de programação dinâmica, funcional e homoicônica construída em cima da máquina virtual Erlang com um poderoso sistema de macro que a torna ideal para a construção de Linguagens Específicas de Domínio (DSLs). O Elixir tem características distintivas, tais como o operador Pipe que permite a construção de um *pipeline* de funções como você faria na linha de comando shell do UNIX. O *bytecode* compartilhado permite que o Elixir interopere com o Erlang e reaproveite bibliotecas existentes, além de fornecer ferramentas de apoio, como a ferramenta de *build* Mix, o shell interativo Iex e o framework de testes unitários ExUnit. Ele é uma alternativa prática ao Erlang para a construção de DSLs.

Julia é uma linguagem de programação dinâmica, procedural e homoicônica, projetada para atender às necessidades da computação científica de alto desempenho. A implementação da linguagem é organizada em torno do conceito de funções genéricas e despache dinâmico de métodos. Programas Julia são em sua maioria funções que podem conter várias definições para diferentes combinações de tipos de argumento. A combinação desses recursos de linguagem e o compilador *just-in-time* baseado no LLVM permite que Julia alcance um alto nível de desempenho. Ela também suporta um ambiente de multiprocessamento baseado em passagem de mensagens que permite que programas sejam executados em vários processos. Isso permite aos programadores criarem programas distribuídos baseados em qualquer modelo de programação paralela.

O PowerShell continua sendo uma opção bastante utilizada para fazer automação de baixo nível em máquinas Windows. O **Pester** é uma biblioteca de teste que possibilita executar e validar comandos PowerShell. O Pester simplifica o teste de scripts durante o desenvolvimento com um poderoso sistema de *mocking*, que permite a configuração de *stubs* e *mocks* em testes. Os testes do Pester também podem ser integrados com um sistema de integração contínua para evitar defeitos de regressão.

O **Python 3** foi uma grande mudança, introduzindo alterações que quebraram a compatibilidade com a versão Python 2.x. Ele foi notável por remover recursos de linguagem com o objetivo de torná-lo mais fácil de usar e mais consistente, sem reduzir o seu poder. Isso levou a problemas na adoção, já que algumas das bibliotecas de suporte não foram portadas, e os desenvolvedores Python muitas vezes precisam encontrar novas maneiras de fazer as mesmas coisas. Apesar de tudo, a tentativa de tornar a linguagem mais simples deve ser aplaudida e se você está desenvolvendo ativamente em Python, dê mais uma olhada no Python 3.

Depois de alguns atrasos, causados principalmente por pedidos de patentes da Apple, o W3C finalizou a recomendação do Touch Events. No entanto, nesse meio tempo, o **Pointer Events**, um padrão mais novo, mais amplo e mais rico, está pegando ritmo. Recomendamos considerar o Pointer Events para interfaces HTML que precisam funcionar em diferentes dispositivos.

O **TypeScript** é uma abordagem interessante para introdução de uma nova linguagem de programação para o navegador. Com o TypeScript, os novos recursos da linguagem compilam para JavaScript normal e, mesmo sendo um superconjunto de JavaScript, não parece uma linguagem completamente nova. Ele não representa uma proposição isso-ou-aquilo e não relega o JavaScript para uma plataforma de execução intermediária. Muitos dos recursos da linguagem são baseados em futuras extensões planejadas para o JavaScript.

O **Yeoman** tenta tornar os desenvolvedores de aplicações web mais produtivos simplificando atividades como *scaffold*, *build* e gerenciamento de pacotes. Ele é uma coleção das ferramentas Yo, Grunt e Bower que funcionam bem em conjunto.

Nós continuamos vendo equipes enfrentando problemas com o **JSF** – JavaServer Faces – e recomendamos evitar essa tecnologia. As equipes parecem escolher o JSF por ser um padrão J2EE, sem realmente avaliar se o modelo de programação é conveniente. Achamos que o JSF possui falhas porque tenta abstrair o HTML, o CSS e o HTTP, exatamente o oposto do que fazem os frameworks web modernos. O JSF, assim como o ASP.NET webforms, tenta criar estado persistente em cima do protocolo HTTP, que não possui estado, o que acaba gerando uma série de problemas envolvendo o estado compartilhado do lado do servidor. Nós estamos conscientes das melhorias no JSF 2.0, mas achamos que o modelo é fundamentalmente falho. Recomendamos que as equipes utilizem frameworks simples, abracem e compreendam as tecnologias da web, incluindo o HTTP, o HTML e o CSS.

REFERÊNCIAS

Interações Tangíveis
http://www.interaction-design.org/encyclopedia/tangible_interaction.html
http://www.computer.org/csdl/mags/co/2013/08/mco2013080070-abs.html
http://www.theverge.com/2012/9/21/3369616/co-working-robots-baxter-home
http://robohub.org/rethink-robotics-baxter-and-universal-robots-ur5-and-ur10-succeeding/

Padrão de Componentes Web
http://www.polymer-project.org

Hystrix
https://github.com/Netflix/Hystrix/wiki
https://github.com/Netflix/Hystrix/tree/master/hystrix-dashboard
https://github.com/Netflix/Turbine/wiki

Extensões Reativas para Várias Linguagens
https://github.com/blog/1107-reactivecocoa-for-a-better-world
http://facebook.github.io/react/
http://techblog.netflix.com/2013/02/rxjava-netflix-api.html
http://elm-lang.org/
http://www.flapjax-lang.org/

Pointer Events
http://www.w3.org/TR/pointerevents/
http://www.w3.org/TR/touch-events/
http://www.w3.org/2012/te-pag/pagreport.html
http://msopentech.com/blog/2013/06/17/w3c-pointer-events-gains-further-web-momentum-with-patch-for-mozilla-firefox

Sobre a ThoughtWorks – É uma consultoria global, empresa de produtos de software e uma comunidade de pessoas apaixonadas cujo propósito é revolucionar o desenvolvimento e criação de software, promovendo impacto social positivo nos países e comunidades onde atua. Sua divisão de produtos, a ThoughtWorks Studios, desenvolve ferramentas pioneiras para equipes de software - tais como Mingle ®, Go ™ e Twist ®, e que ajudam as organizações a colaborar e entregar software de qualidade. Os clientes da ThoughtWorks são organizações com missões ambiciosas que buscam abordagens e tecnologias inovadoras como forma de atingir seus objetivos. Com 20 anos de atuação no mercado, a ThoughtWorks tem mais de 2.500 funcionários – os 'ThoughtWorkers' – atendendo clientes em escritórios na África do Sul, Alemanha, Austrália, Brasil, Canadá, China, Estados Unidos, Equador, Índia, Inglaterra, Singapura e Uganda.

CONTRIBUIDORES - O TAB da ThoughtWorks é composto por:

Rebecca Parsons (CTO)
Martin Fowler (Cientista Chefe)
Badri Janakiraman
Brain Leke
Claudia Melo
Darren Smith
Erik Doernenburg
Evan Bottcher
Hao Xu
Ian Cartwright
James Lewis
Jeff Norris
Jonny LeRoy
Mike Mason
Neal Ford
Rachel Laycock
Sam Newman
Scott Shaw
Srihari Srinivasan
Thiyagu Palanisamy