JANUARY 2014

TECHNOLOGY RADAR

Prepared by the ThoughtWorks Technology Advisory Board

thoughtworks.com/radar

ThoughtWorks®

WHAT'S NEW?

Here are the trends highlighted in this edition:

- **Early warning and recovery in production** - We are seeing a plethora of new tools and techniques for logging, monitoring, storing and querying operational data. When combined with the short recovery times afforded by virtualization and infrastructure automation, businesses can reduce the amount of testing required before deployment, perhaps even pushing that testing into the production environment itself.

- **Privacy vs. big data** - While we are excited about the new business insights made possible by exhaustive data collection and the new tools and platforms for storing and analyzing that data, we are also concerned that many businesses are storing vast amounts of personal data unnecessarily. We advocate that businesses adopt an attitude of "datensparsamkeit" and store only the absolute minimum personal information from their customers.

- **The Javascript juggernaut rolls on** - The ecosystem around JavaScript as a serious application platform continues to evolve. Many interesting new tools for testing, building, and managing dependencies in both server- and client-side JavaScript applications have emerged recently.

- **Merging of physical and digital** - Low-cost devices, open hardware platforms, and new communication protocols are pushing the computing experience away from the screen and into the world around us. A great example is the proliferation of wearable devices to track personal biometrics, and hardware support in mobile devices to interact with these devices.

ThoughtWorkers are passionate about technology. We build it, research it, test it, open source it, write about it, and constantly aim to improve it – for everyone. Our mission is to champion software excellence and revolutionize IT. We create and share the ThoughtWorks Technology Radar in support of that mission. The ThoughtWorks Technology Advisory Board, a group of senior technology leaders in ThoughtWorks, creates the radar. They meet regularly to discuss the global technology strategy for ThoughtWorks and the technology trends that significantly impact our industry.

The radar captures the output of the Technology Advisory Board's discussions in a format that provides value to a wide range of stakeholders, from CIOs to developers. The content is intended as a concise summary. We encourage you to explore these technologies for more detail. The radar is graphical in nature, grouping items into techniques, tools, platforms, and languages & frameworks. When radar items could appear in multiple quadrants, we chose the one that seemed most appropriate. We further group these items in four rings to reflect our current position on them. The rings are:

- **Adopt:** We feel strongly that the industry should be adopting these items. We use them when appropriate on our projects.
- **Trial:** Worth pursuing. It is important to understand how to build up this capability. Enterprises should try this technology on a project that can handle the risk.
- **Assess:** Worth exploring with the goal of understanding how it will affect your enterprise.
- **Hold:** Proceed with caution.

Items that are new or have had significant changes since the last radar are represented as triangles, while items that have not moved are represented as circles. The detailed graphs for each quadrant show the movement that items have taken. We are interested in far more items than we can reasonably fit into a document this size, so we fade many items from the last radar to make room for the new items. Fading an item does not mean that we no longer care about it.

For more background on the radar, see **http://martinfowler.com/articles/radar-faq.html**

CONTRIBUTORS - The ThoughtWorks Technology Advisory Board is comprised of:

Rebecca Parsons (CTO)
Martin Fowler (Chief Scientist)
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

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

THE RADAR

TECHNIQUES

ADOPT
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
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
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
22 Cloud lift and shift
23 Ignoring OWASP Top 10
24 Siloed metrics
25 Velocity as productivity

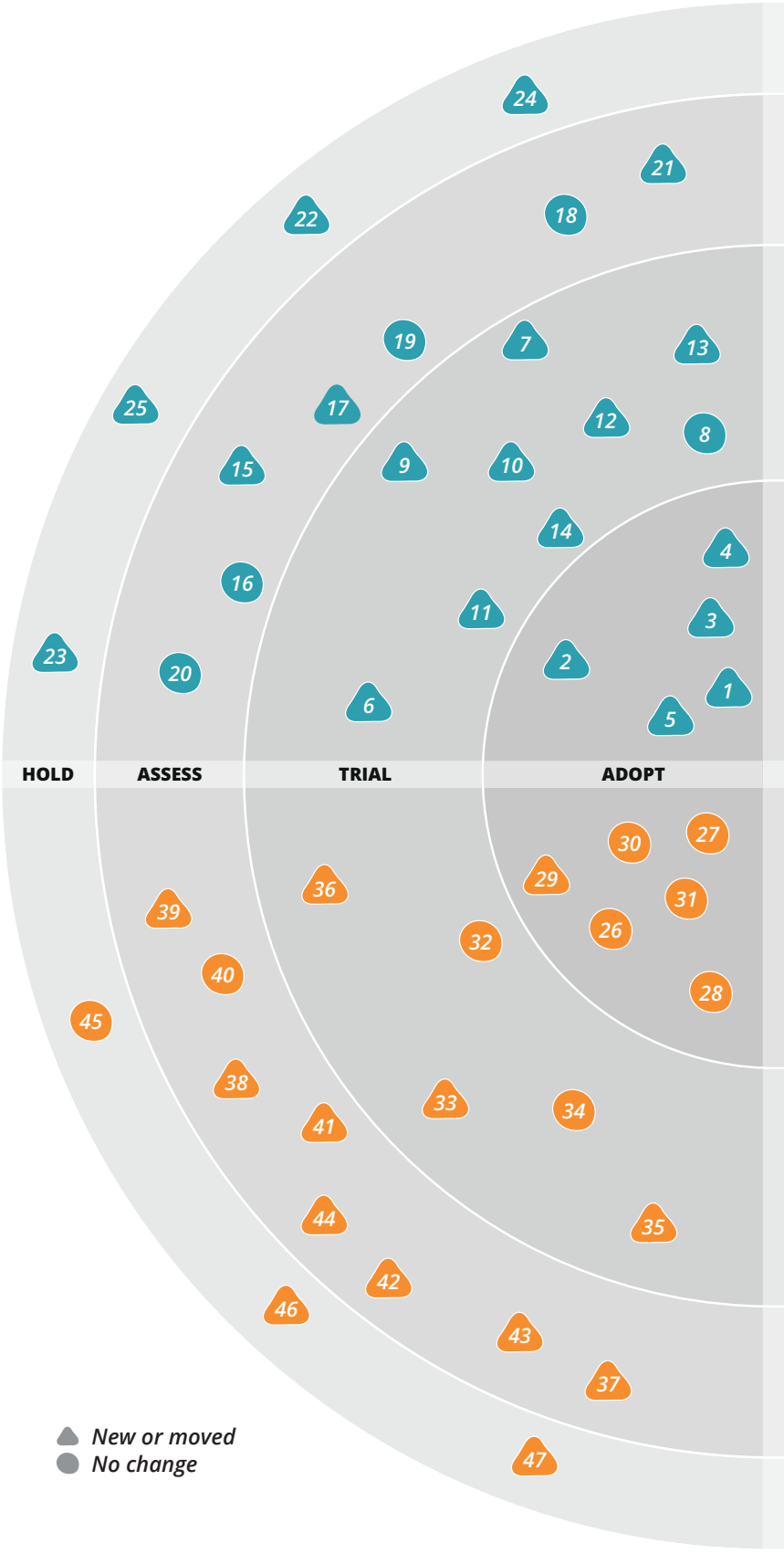
PLATFORMS

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

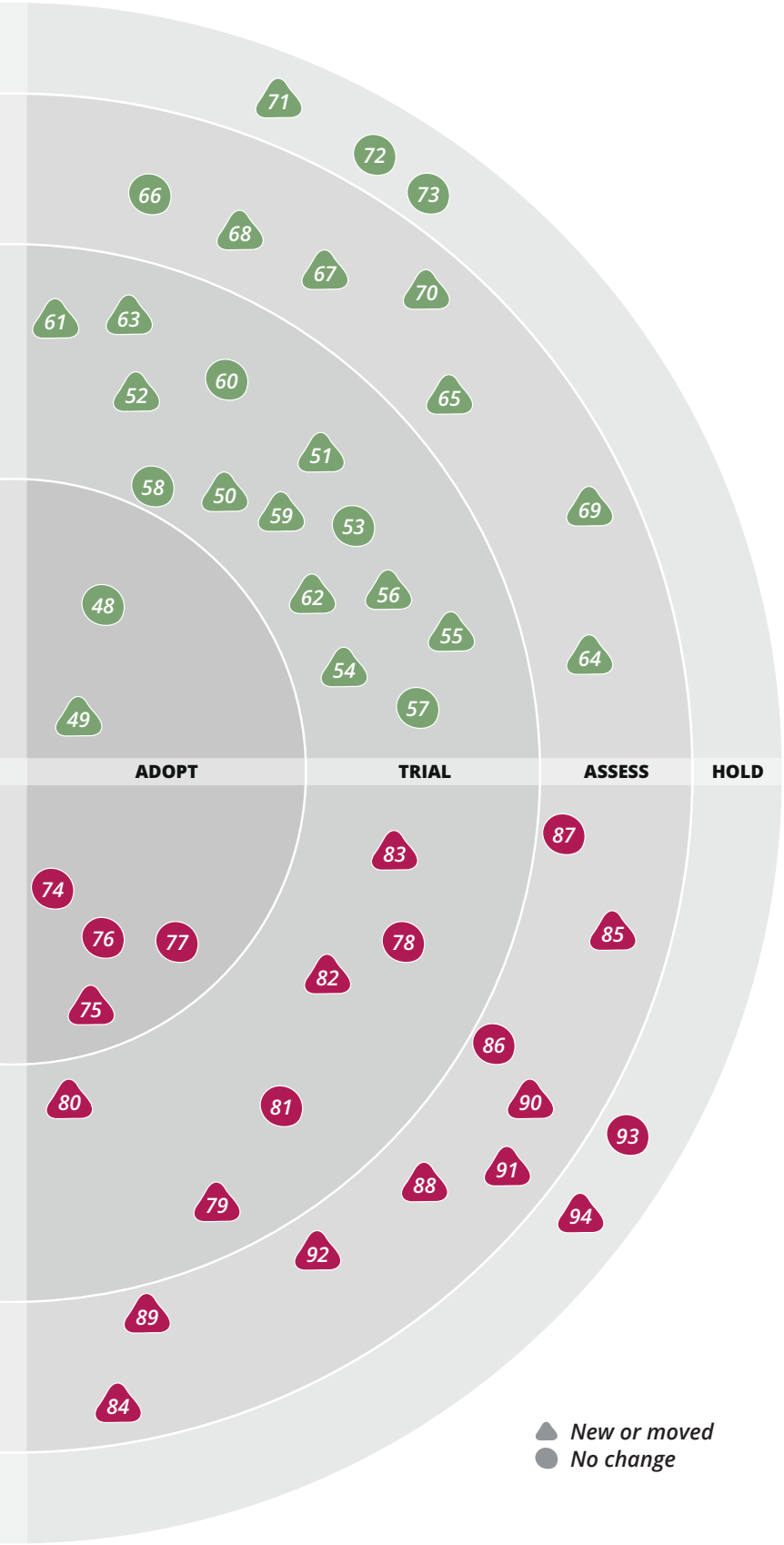
TRIAL
32 Hadoop 2.0
33 Hadoop as a service
34 OpenStack
35 PostgreSQL for NoSQL
36 Vumi

ASSESS
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
45 Big enterprise solutions
46 CMS as a platform
47 Enterprise Data Warehouse



© January 2014, ThoughtWorks, Inc. All Rights Reserved.



THE RADAR

TOOLS

ADOPT
48 D3
49 Dependency management for JavaScript

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

HOLD
71 Ant
72 Heavyweight test tools
73 TFS

LANGUAGES & FRAMEWORKS

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

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

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

HOLD
93 Handwritten CSS
94 JSF

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

TECHNIQUES

Capturing client-side JavaScript errors has helped our delivery teams identify issues specific to a browser or plug-in configuration that impact user experience. Over the past year a number of service providers have started to surface in support of this requirement. Other than storing these errors in application data stores, web applications can log this data to web analytics or existing monitoring tools such as New Relic to offload storage requirements.

Since the last radar a few advances have made **continuous delivery** for native apps on **mobile devices** less painful. Xctool, the recently open-sourced "better xcodebuild" improves iOS build automation and unit testing. The arrival of automatic updates in iOS7 reduces the friction of regular releases. Travis-CI now supports OS X agents, removing another hurdle in seamless CD pipelines for mobile platforms. Our advice from the last radar on the value of hybrid approaches and the importance of test automation for mobile still applies.

As client-side JavaScript applications grow in sophistication, we see an increased need for engineering sophistication to match. A common architectural flaw is unfettered access to the DOM from across the codebase - mixing DOM manipulation with application logic and AJAX calls. This makes the code difficult to understand and extend. Thinking about separation of concerns is a useful antidote. This involves aggressively restricting all DOM access (which usually translates to all jQuery usage) to a thin 'segregation layer'. One pleasant side-effect of this approach is that everything outside of that **segregated DOM** layer can be tested rapidly in isolation from the browser using a lean JavaScript engine such as **node.js.**

When using techniques such as "instrument all the things" and semantic logging, it can be very useful to **capture domain events explicitly**. You can avoid having to infer user intent behind state transitions by modeling these transitions as first-class concerns. One method of achieving this outcome is to use an event sourced architecture with application events being mapped to business meaningful events.

Increasingly, HTML is rendered not only on the server but also on the client, in the web browser. In many cases this split rendering will remain a necessity but with the growing maturity of JavaScript templating libraries an interesting approach has become viable: **client and server rendering with same code.**

You cannot act on important business events unless you monitor them. The principle, **instrument all the things**, encourages us to think proactively about how we achieve this at the start of our software development. This allows us to expose key metrics, monitor them, and report on them to improve operational effectiveness.

Chef & Puppet servers are a central place to store recipes/manifests that propagate configuration changes to managed machines. They are also a central database of node information and provide access control for manifests/recipes. The disadvantage of having these servers is that they become a bottleneck when multiple clients simultaneously connect to them. They are a single point of failure and take effort to be robust and reliable. In light of this, we recommend **chef-solo or standalone puppet** in conjunction with a version control system when the server is primarily used to store recipes/manifests. Teams can always introduce the servers as the need arises or if they find themselves reinventing solutions to the problems the servers have already solved.

Increasingly we are unbounded by our ability to procure and provision hardware. However with the massive increase in flexibility this affords us, we have found that we are bounded

ADOPT
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
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
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
22 Cloud lift and shift
23 Ignoring OWASP Top 10
24 Siloed metrics
25 Velocity as productivity

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

TECHNIQUES *continued*

by the scale and complexity of the software assets used to manage our virtual estates. Using techniques more familiar in the software development world such as TDD, BDD and CI offers an approach to managing this complexity and gives us the confidence to make changes to our infrastructure in a safe, repeatable and automatable manner. **Provisioning testing** tools, like rspec-puppet, Test Kitchen and serverspec, are available for most platforms.

Treating logs as data gives us greater insight into the operational activity of the systems we build. **Structured logging**, which is using a consistent, predetermined message format containing semantic information, builds on this technique and enables tools such as Graylog2 and Splunk to yield deeper insights.

The reduction in cost, size, power consumption and simplicity of physical devices has led to an explosion in devices that open physical domains to software. These devices often contain little more than a sensor and a communication component like Bluetooth Low Energy or WiFi. As software engineers, we need to expand our thinking to include **bridging physical and digital worlds with simple hardware**. We are already seeing this in the car, the home, the human body, agriculture and other physical environments. The cost and time required to prototype such devices is shrinking to match the fast iterations possible in software.

In our desire to support ever-changing business models, learn from past behavior and provide the best experience for every individual visitor, we are tempted to record as much data as possible. At the same time hackers are more ferocious than ever, with one spectacular security breach after another, and we now know of unprecedented mass-surveillance by government agencies. The term **Datensparsamkeit** is taken from German privacy legislation and describes the idea to only store as much personal information as is absolutely required for the business or applicable laws. Some examples are instead of storing a customer's full IP address in access logs, just using the first two or three octets and instead of logging transit journeys with a username using an anonymous token. If you never store the information, you do not need to worry about someone stealing it.

As the lines between hardware and software continue to blur, we see traditional computing increasingly embedded in everyday objects. Although connected devices are now ubiquitous in retail spaces, automobiles, homes, and workplaces, we still do not understand how to blend them into a useful computing experience that goes beyond a simple glass screen. **Tangible interaction** is a discipline that blends software and hardware technology, architecture, user experience, and industrial design. The goal is to provide natural environments made up of physical objects where humans can manipulate and understand digital data.

As cloud adoption grows we are unfortunately seeing a trend to treat the cloud as just another hosting provider. This **cloud lift and shift** trend is unfortunately being encouraged by large vendors re-branding existing hosting offerings as "cloud." Few of these offer any real flexibility or pay-as-you-use pricing. If you think you can move to the cloud without re-architecting you are probably not doing it right.

Barely a week goes by without the IT industry being embarrassed by yet another high profile loss of data, leak of passwords, or breach of a supposedly secure system. There are good resources to help with making sure security gets treated as a first-class concern during software development and we need to stop ignoring them; the **OWASP Top 10** is a good place to start.

As more businesses move online we have noted a tendency to end up with **siloed metrics**. Specific tools are implemented to gather and display specific metrics: one tool for page-views and browser behavior, another for operational data and another to consolidate log messages. This leads to data silos and the need to swivel-chair integrate between the tools in order to gather business intelligence that is crucial to running the business. This is a tool-led split in the analytics domain that hurts the team's ability to make decisions. A much better solution is to have a consolidated view of near-real time analytics using integrated dashboards displaying time-sensitive domain and team relevant information.

Of all the approaches that we might disagree with, equating velocity with productivity has become so prevalent that we felt it important to call it out in our hold ring. When properly used, velocity allows the incorporation of "yesterday's weather" into the iteration planning process. Velocity is simply a capacity estimate for a given team at a given time. It can improve as a team gels or by fixing problems like technical debt or a flaky build server. However, like all metrics, it can be misused. For example, over-zealous project managers attempt to insist on continual improvement of velocity. **Treating velocity as productivity** leads to unproductive team behaviors that optimize the metric at the expense of actual working software.

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

TOOLS

Using **Dependency management** tools for **JavaScript** has helped our delivery teams handle large amounts of JavaScript by structuring their code and loading the dependencies at runtime. Though this simplified the effort in most cases, lazy loading complicates supporting offline mode. Different dependency management tools have different strengths, so choose based on your context.

In the category of DevOps orchestration engines, **Ansible** has nearly universal acclaim within ThoughtWorks projects. It has useful tools and abstractions at a useful level of granularity.

On mobile projects, we have been impressed with the functionality and gradually evolving capabilities and maturity of **Calabash**. It is an automated acceptance test tool for both Android and iOS applications that supports common ecosystem tools like Cucumber. It is an attractive choice on heterogeneous projects.

Following our recommendation in the last radar to consider a focus on reducing mean time to recovery, we want to highlight **Chaos Monkey** from Netflix's Simian Army suite. It is a tool that randomly disables instances in the production environment during normal operation. When run with comprehensive monitoring and a team on stand by, it helps to uncover unexpected weaknesses in the system, which in turn allows the development team to build automatic recovery mechanisms ahead of time, rather than struggling to respond to an outage that caught everyone by surprise.

Several of our ThoughtWorks teams developing Node.js apps are using **Grunt** to automate most of the development activities like minification, compilation, and linting. Many of the common tasks are available as Grunt plugins. You can even programmatically generate the configuration if necessary.

Managing the web of dependencies in a distributed system is complicated, and is a problem more people are facing with the move to finer-grained micro-services. **Hystrix** is a library for the JVM from Netflix that implements patterns for dealing with downstream failure, offers real-time monitoring of connections, and caching and batching mechanisms to make inter-service dependencies more efficient. In combination with hystrix-dashboard and Turbine, this tool can be used to build

more resilient systems and provide near-real time data on throughput, latency and fault tolerance.

Testing HTTP-based micro-services can be painful and tricky. Particularly in two scenarios, the consumption of a group of micro-services from front-end, and the communication between micro-services. To deal with these, **Moco** can be handy. It is a lightweight stub framework for testing HTTP-based endpoints. You can have an embedded stubbed service up and running with 2 lines of Java or Groovy code, or a standalone one with few lines of JSON to describe the required behavior.

We have long favored the use of hand-drawn, low fidelity prototypes to illustrate user interactions without getting caught up in the nitty-gritty of the graphic design. **Prototype On Paper** is a tool that allows individual mockups drawn on paper to be captured via camera on iOS or Android and linked together to allow for testing of user interaction. This bridges the gap nicely between the static, lo-fi paper prototypes and more hi-fi prototyping techniques.

ADOPT
48 D3
49 Dependency management for JavaScript

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

HOLD
71 Ant
72 Heavyweight test tools
73 TFS

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

TOOLS *continued*

We mentioned ThoughtWorks' **SnapCI** -- a hosted service that provides deployment pipelines -- on the last edition of the Radar. Since then, we have seen many teams successfully use SnapCI on their projects. If you need a simple continuous delivery solution in the cloud, SnapCI can provide it with one click. No hardware, no hassle.

With increasing scrutiny over the privacy of data, more companies are concerned about sharing web analytics with third parties. **Snowplow Analytics and Piwik** are examples of open-source analytics platforms that can be self-hosted and provide a promising feature set and roadmap.

Cloud-init is a simple but powerful technique for carrying out actions on a cloud instance at boot time. It is particularly useful when used with instance metadata to allow a newly booted instance to pull the configuration, dependencies and software needed to perform a particular role. When used together with the Immutable or Phoenix server pattern, this can create a very responsive and light-weight mechanism for managing deployments in the cloud.

The **Docker** open-source project has generated a great deal of interest within ThoughtWorks, and is growing in momentum and maturity. Docker allows applications to be packaged and published as portable lightweight containers that run identically on a laptop or a production cluster. It provides tooling for the creation and management of application containers, and a run-time environment based on LXC (LinuX Containers).

Many monitoring tools are built around the idea of the machine. We monitor what the machine is doing and which software is running on it. When it comes to cloud based infrastructure, especially patterns like Phoenix and Immutable servers this is a problematic approach. Machines come and go, but what is important is that the services remain working. **Sensu** allows a machine to register itself as playing a particular role and Sensu then monitors it on that basis. When we are finished with the machine we can simply de-register it.

All development for iOS must be carried out on OS X. Due to technical and licensing restrictions running server farms with OS X is neither easy nor common. In spite of these difficulties, **Travis CI**, with support from Sauce Labs, now provides cloud-based continuous integration services for iOS and OS X projects.

Growing complexity in web applications has increased the awareness that appearance should be tested in addition to functionality. This has given rise to a variety of **visual regression testing tools**, including CSS Critic, dpxdt, Huxley, PhantomCSS, and Wraith. Techniques range from straightforward assertions of CSS values to actual screenshot comparison. While this is a field still in active development we believe that testing for visual regressions should be added to continuous delivery pipelines.

Among the various choices available for building cross-platform mobile apps, **Xamarin** offers a fairly unique toolset. It supports C# and F# as the primary language with bindings to platform specific SDKs and the Mono runtime environment that works across iOS, Android and Windows Phone. Applications are compiled to native code instead of the typical cross-platform approach that renders HTML-based UI in an embedded browser. This gives apps a more native look and feel. When using this toolset, it is imperative that the platform specific UI tier be separated from the rest of the tiers to ensure code reuse across different platforms. The application binary tends to be a bit bigger due to the runtime environment that is included.

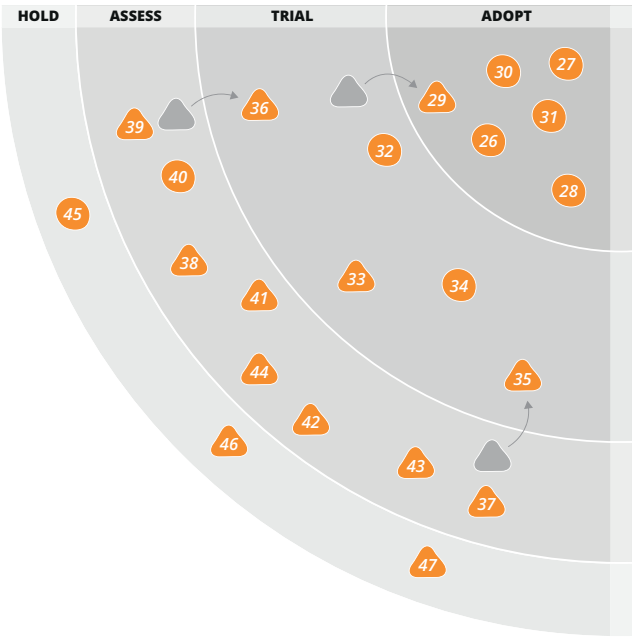
We continue to see teams expend significant effort on un-maintainable **Ant** and Nant build scripts. These are hard to understand and extend due to the inherent lack of expressiveness and clean modularity provided by the tools. Alternatives like Gradle, Buildr, and PSake have clearly demonstrated superior maintainability and productivity.

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

PLATFORMS

We observe organizations that have piloted Hadoop successfully starting to consolidate their Hadoop infrastructure services into a centralized, managed platform before rolling it out across the enterprise. These **Hadoop-as-a-Service** platforms are characterized by the control tier that interfaces with and coordinates among different core Hadoop infrastructure components. The capabilities of the platform are usually exposed via higher-level abstractions to the enterprise. Such a managed platform gives organizations the ability to deploy processes, infrastructure and datasets in a fairly consistent way across the organization. These services are built in private data centers and public cloud infrastructure.

Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM. It offers very lightweight event-driven processes with approximately 2.7 million actors per GB RAM and a "let-it-crash" model of fault-tolerance designed to work in a distributed environment. Akka can be used as a library for web-apps or as a stand-alone kernel to drop an application into.



The recent explosion of mobile-focused products, coupled with widespread adoption of "Lean Start-up" approaches that put a premium on time-to-market for new ideas, has spawned an ecosystem of **Backend-as-a-service** (BaaS) offerings that enable developers to focus on the client application while offloading backend concerns. Assess adding these services to your toolkit where fast and low-cost proving of a new product idea is important. Our usual advice on build/buy/borrow decisions still applies: be clear on which functional areas are strategic to your business and which are commodities. For potentially strategic areas be sure to plan a migration path that will allow you to use the BaaS provider to get started quickly, while avoiding friction when your architecture evolves and you need to migrate to owning this functionality and customizing it as a differentiator.

With the cost of industrial robots dropping and their safety and ease of use increasing, the world of useful, commercial robotics is opening up. Robots like Rethink Robotics' Baxter* or Universal Robotics' U5, make it feasible for small to medium-sized businesses to automate repetitive tasks previously performed by humans. Increasingly, enterprise software will have to integrate with **low-cost robotics** as another participant in the value stream. The challenge lies in making the experience easy and productive for the human co-workers as well.

The need for physically storing data within nations or organizations has increased significantly in recent years. There is concern around sensitivity of information hosted in cloud environments. Organizations are looking into **private cloud** as an alternative when data that needs to be housed in close proximity with control over access and distribution. Private cloud offers cloud infrastructure provisioned for exclusive use by a single organization with the following characteristics; on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service.

SPDY is an open networking protocol for low-latency transport of web content proposed for HTTP2 that has seen a rise in modern browser support. SPDY reduces page load time by prioritizing the transfer of subresources so that only one

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

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

ASSESS
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
45 Big enterprise solutions
46 CMS as a platform
47 Enterprise Data Warehouse

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

PLATFORMS *continued*

connection is required per client. Transport layer security is used in SPDY implementations with the transmission headers gzip or deflate compressed instead of human-readable text in HTTP. It is great for high-latency environments.

Heterogeneous and overwhelmingly large amounts of data is not the only theme of big data. In certain circumstances, speed of processing can be as important as the volume. **Storm** is a distributed realtime computation system. It has similar scalability to Hadoop, with throughput as fast as a million tuples per second. It enables for real time processing what Hadoop does for batch.

In the previous radar we cautioned against the use of traditional web component frameworks that provide a component model on the server side. The **Web Components standard** that originated at Google, is something quite different. It provides an easier way to create recyclable widgets by helping with encapsulation of HTML, CSS and JavaScript, so they do not interfere with the rest of the page and the page does not interfere with them. Developers can use as much or as little of the framework as needed. Early support is provided by the Polymer Project.

While centralized integration of data for analysis and reporting remains a good strategy, traditional **Enterprise Data Warehouse** (EDW) initiatives have a higher than 50% failure rate. Big up-front data modeling results in overbuilt warehouses that take years to deliver and are expensive to maintain. We are placing these old-style EDWs and techniques on hold in this edition of the radar. Instead, we advocate evolving towards an EDW. Test and learn by building small, valuable increments that are frequently released to production. Nontraditional tools and techniques can help, for example using a Data Vault schema design or even a NoSQL document store such as HDFS.

Content Management Systems (CMS) have their place. In many cases it is unreasonable to write editing and workflow functionality from scratch. However, we have experienced serious problems when **CMS-as-a-platform** becomes an IT solution that grows beyond managing simple content.

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

LANGUAGES & FRAMEWORKS

Scala is a large language that is popular because of its approachability for new developers. This banquet of features is a problem because many aspects of Scala, like implicit conversions and dynamics, can get you into trouble. To successfully use Scala, you need to research the language and have a very strong opinion on which parts are right for you, creating your own definition of **Scala, the good parts**. You can disable the parts you do not want using a system called feature flags.

The **Go language** was originally developed by Google as a system programming language to replace C & C++. Four years out, Go is gaining traction in other areas. The combination of very small, statically linked binaries combined with an excellent HTTP library means Go has been popular with organizations making use of finer-grained, micro-service architectures.

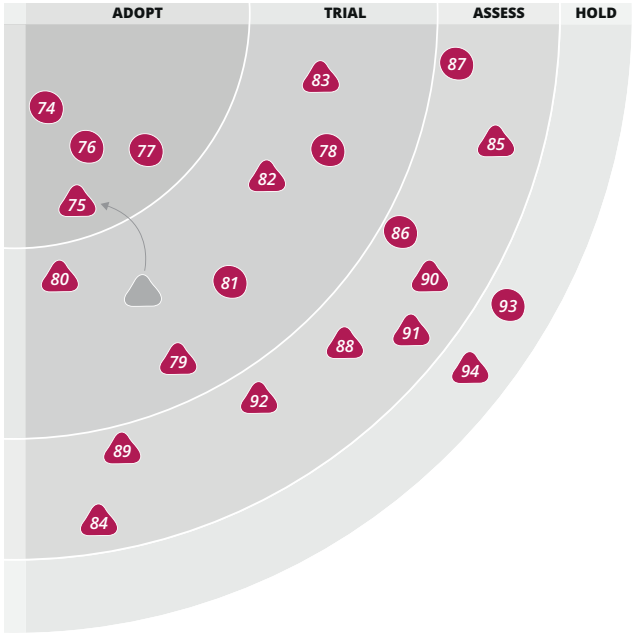
Hive is a data warehouse built on top of Hadoop which provides a SQL-like query and data definition language that converts queries into MapReduce jobs that can be run across the entire Hadoop cluster. Like all useful abstractions, Hive does not try to deny the existence of the underlying mechanics of Hadoop and supports custom map-reduce operations as a powerful extension mechanism. Despite the superficial similarities to SQL, Hive does not try to be a replacement for low-latency, real-time query engines found on relational database systems. We strongly advise against using Hive for online ad-hoc querying purposes.

The **Play Framework 2** blip has generated many internal discussions. We had competing suggestions to move it to adopt and hold. These differences relate primarily to the specific applications for which it is used, how it is used, and what expectations people have for it. While none of these issues are unique for Play, Play has generated far more controversy than is typical in the standard library versus framework debate. We reiterate the cautions stated in the previous radar, and we will monitor how Play continues to mature to support its sweet spot.

Reactive Programming deals with streams or values that change over time. Using elements of data flow, implicit concurrency and transparent event propagation, these techniques enable efficient handling of events on a large scale with a high degree of efficiency and low latency. In the previous radar, we mentioned Reactive Extensions in .NET due to the extensive work done by Microsoft in making Rx a core part of the .NET framework. Since then, with the introduction of the Reactive Cocoa library for Objective C, the Java port of Reactive Extensions, the React JavaScript library, the Elm language based on Haskell & the Flapjax JavaScript library, we are extending this blip to include **Reactive Extensions across languages**.

Until recently, Microsoft's **Web API** was the least-worst option for building a RESTful service using ASP.NET. Web API 2 fixes a number of rough edges with better support for flexible routing, sub-resources, media types and improved testability. It continues to be our preferred library for building .NET REST APIs.

Elixir is a dynamic, functional, homoiconic programming language built on top of the Erlang virtual machine with a powerful macro system that makes it ideal for building Domain Specific Languages. Elixir has distinctive features such as the Pipe operator that allows developers to build a pipeline of functions like you would in the UNIX command



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

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

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

HOLD
93 Handwritten CSS
94 JSF

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

LANGUAGES & FRAMEWORKS *continued*

shell. The shared byte code allows Elixir to interoperate with Erlang and leverage existing libraries while supporting tools such as the Mix build tool, the Iex interactive shell and the ExUnit unit testing framework. It is a practical alternative to Erlang for building DSLs.

Julia is a dynamic, procedural and homoiconic programming language designed to address the needs of high performance scientific computing. The implementation of the language is organized around the concept of generic functions and dynamic method dispatch. Julia programs are largely functions that can contain multiple definitions for different combinations of argument types. The combination of these language features and the LLVM based just-in-time compiler help Julia achieve a high level of performance. Julia also supports a multiprocessing environment based on message passing to allow programs to run on multiple processes. This enables programmers to create distributed programs based on any of the models for parallel programming.

PowerShell remains a widely used option for doing low-level automation on Windows machines. **Pester** is a testing library that makes it possible to execute and validate PowerShell commands. Pester simplifies testing of scripts during development with a powerful mocking system that makes it possible to setup stubs and doubles in tests. Pester tests can also be integrated into a continuous integration system to prevent regression defects.

Python 3 was a major change from the previous Python 2.x that introduced backwards incompatible changes. It was notable for actually removing languages features to make it easier to use and more consistent, without reducing its power. This has led to problems in adoption, as some of the supporting libraries people rely on have not been ported, and Python developers often have to find new ways of doing things. Nonetheless the drive towards making a language simpler is to be applauded, and if you are actively developing in Python, then give Python 3 another look.

After some delays, mainly caused by patent claims from Apple, the W3C has now finalized the Touch Events recommendation. However, in the meantime, **Pointer Events**, a newer, broader, and richer standard, is picking up momentum. We recommend considering Pointer Events for HTML interfaces that must work across different input methods.

TypeScript is an interesting approach to bringing a new programming language to the browser. With TypeScript, the new language features compile down to normal JavaScript, and yet as a superset of JavaScript it does not feel like a completely new language. It does not represent an either-or proposition and it does not relegate JavaScript to an intermediate execution platform. Many of the language features are based on planned future extensions of JavaScript.

Yeoman attempts to make web application developers more productive by simplifying activities like scaffold, build and package management. It is a collection of the tools Yo, Grunt and Bower that work well as a set.

We continue to see teams run into trouble using **JSF** -- JavaServer Faces -- and are recommending you avoid this technology. Teams seem to choose JSF because it is a J2EE standard without really evaluating whether the programming model suits them. We think JSF is flawed because it tries to abstract away HTML, CSS and HTTP, exactly the reverse of what modern web frameworks do. JSF, like ASP.NET webforms, attempts to create statefulness on top of the stateless protocol HTTP and ends up causing a whole host of problems involving shared server-side state. We are aware of the improvements in JSF 2.0, but think the model is fundamentally broken. We recommend teams use simple frameworks and embrace and understand web technologies including HTTP, HTML and CSS.

© January 2014, ThoughtWorks, Inc. All Rights Reserved.

REFERENCES

Tangible Interactions
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/

Web Components standard
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

Reactive Extensions across languages
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

ThoughtWorks – a software company and community of passionate individuals whose purpose is to revolutionize software creation and delivery, while advocating for positive social change. Our product division, ThoughtWorks Studios, makes pioneering tools for software teams who aspire to be great; such as Mingle®, Go™ and Twist® which help organizations better collaborate and deliver quality software. Our clients are people and organizations with ambitious missions; we deliver disruptive thinking and technology to empower them to succeed. In our 20th year, approximately 2500 ThoughtWorks employees – 'ThoughtWorkers' – serve our clients from offices in Australia, Brazil, Canada, China, Germany, India, Singapore, South Africa, Uganda, the U.K. and the U.S.

© January 2014, ThoughtWorks, Inc. All Rights Reserved.