Technology Radar MAY 2013

Prepared by the ThoughtWorks Technology Advisory Board

thoughtworks.com/radar

ThoughtWorks®

What's New?

Here are the trends highlighted in this edition:

- **Embracing falling boundaries**—Whether you like it or not, boundaries are falling down around you. We choose to embrace this by examining concepts like perimeterless enterprise, development environments in the cloud, and co-location by telepresence.

- **Applying proven practices to areas that somehow missed them**—We are not really sure why, but many in our industry have missed ideas like capturing client side javaScript errors, continuous delivery for mobile, database migrations for NoSQL, and frameworks for CSS.

- **Lightweight options for analytics**—Data science and analytics are not just for people with a PhD in the field. We highlight collaborative analytics and data science, where all developers understand the basics and work closely with experts when necessary.

- **Infrastructure as code**—Continuous delivery and DevOps have elevated our thinking about infrastructure. The implications of thinking about infrastructure as code and the need for new tools are still evolving.

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
Darren Smith
Erik Doernenburg
Evan Bottcher
Hao Xu
Ian Cartwright
James Lewis
Jeff Norris
Mike Mason
Neal Ford
Rachel Laycock
Ronaldo Ferraz
Sam Newman
Scott Shaw
Srihari Srinivasan
Thiyagu Palanisamy

The Radar

Techniques

ADOPT
1 Aggregates as documents
2 Automated deployment pipeline
3 Guerrilla testing
4 In-process acceptance testing
5 Mobile testing on mobile networks
6 Performance testing as a first-class citizen
7 Promises for asynchronous programming
8 Windows infrastructure automation

TRIAL
9 Analyzing test runs
10 Blue-green deployment
11 Co-location by telepresence
12 Continuous delivery for mobile devices
13 Database migrations for NoSQL
14 Edge Side Includes for page composition
15 HTML5 storage instead of cookies
16 Logs as data
17 Micro-services
18 Mobile first
19 Perimeterless enterprise
20 Responsive web design
21 Semantic monitoring

ASSESS
22 Capturing client-side JavaScript errors
23 Collaborative analytics and data science
24 Development environments in the cloud
25 Focus on mean time to recovery
26 Machine image as a build artifact
27 Minimizing application configuration

HOLD
28 Exhaustive browser based testing

Platforms

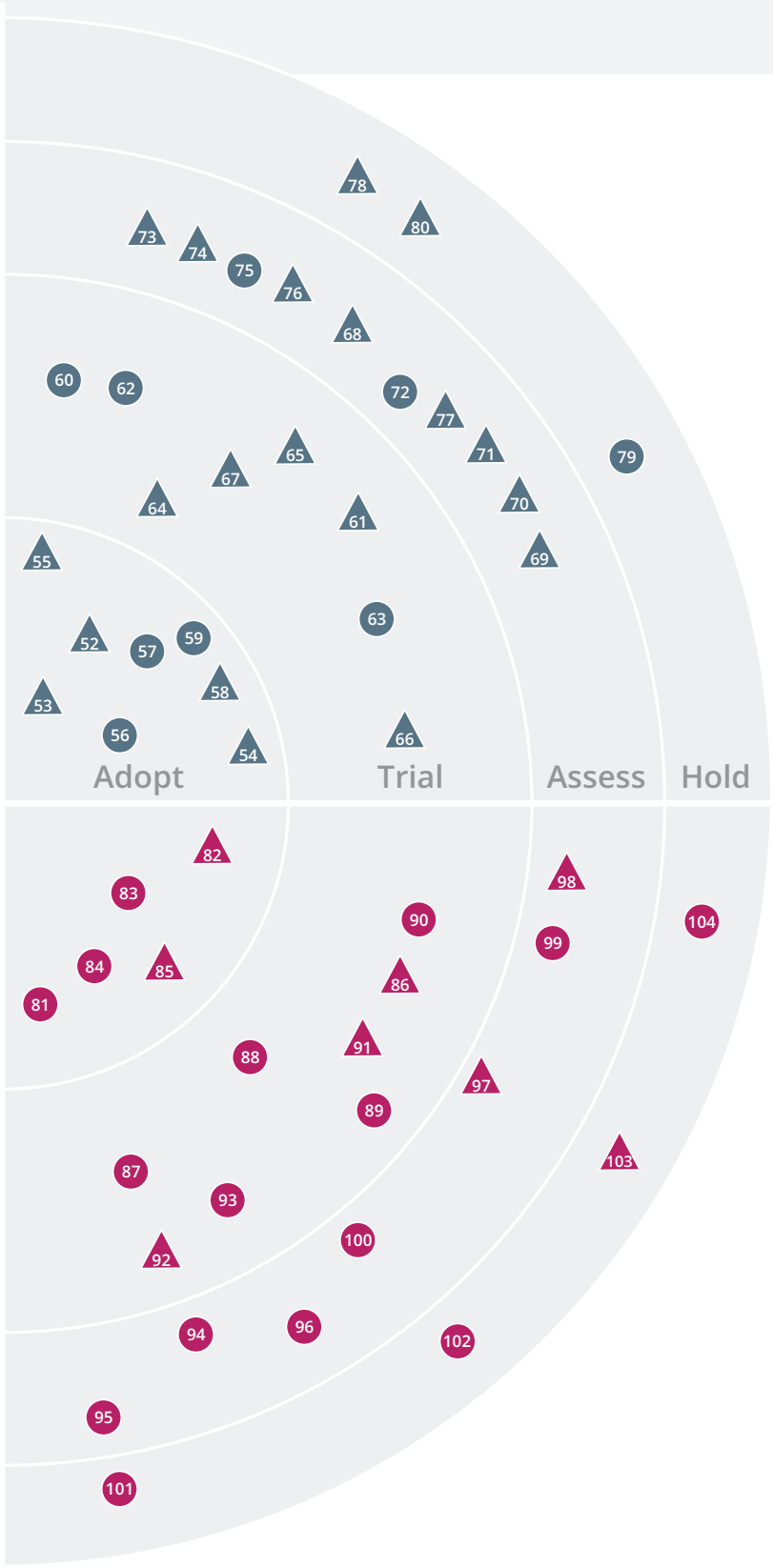
ADOPT
29 Elastic Search
30 MongoDB
31 Neo4J
32 Redis
33 SMS and USSD as a UI

TRIAL
34 BigQuery
35 Continuous integration in the cloud
36 Couchbase
37 Hadoop 2.0
38 Node.js
39 OpenStack
40 Rackspace Cloud
41 Riak

ASSESS
42 Azure
43 Calatrava
44 Datomic
45 PhoneGap/Apache Cordova
46 PostgreSQL for NoSQL
47 Vumi
48 Zepto.js

HOLD
49 Big enterprise solutions
50 Singleton infrastructure
51 WS-*

The Radar



Tools

ADOPT

52 D3
53 Embedded servlet containers
54 Frank
55 Gradle
56 Graphite
57 Immutable servers
58 NuGet
59 PSake

TRIAL

60 Apache Pig
61 Gatling
62 Jekyll
63 Locust
64 Logstash & Graylog2
65 PhantomJS
66 Puppet-librarian and Chef-librarian
67 TestFlight & HockeyApp

ASSESS

68 Browser-based templating
69 Faraday
70 Hystrix
71 Icon fonts
72 Light Table
73 Octopus
74 Reactive Extensions for .Net
75 Riemann
76 Snowplow Analytics
77 UIAutomator

HOLD

78 Heavyweight test tools
79 Maven
80 TFS

Languages & Frameworks

ADOPT

81 Clojure
82 CSS frameworks
83 Jasmine paired with Node.js
84 Scala
85 Sinatra

TRIAL

86 CoffeeScript
87 Dropwizard
88 HTML5 for offline applications
89 JavaScript as a platform
90 JavaScript MV* frameworks
91 Play Framework 2
92 Require.js & NPM
93 Scratch, Alice, and Kodu

ASSESS

94 ClojureScript
95 Gremlin
96 Lua
97 Nancy
98 OWIN
99 RubyMotion
100 Twitter Bootstrap

HOLD

101 Backbone.js
102 Component-based frameworks
103 Handwritten CSS
104 Logic in stored procedures

Techniques

For years, teams and organizations have seen the dangers of siloing expertise around technical disciplines. While we value input from experts on advanced applications, developers should have basic knowledge of user interfaces, databases, and data science, the newest industry darling. While advanced applications requires deep expertise, we are pushing for **collaborative analytics and data science**, where all developers use basic statistical analysis and tools to make better decisions, and work closely with experts when things get complicated.

Technology trends have broken down the garden walls that used to surround corporate IT networks and lead to a **perimeterless enterprise**. Employees frequently use their own consumer devices to access corporate data through cloud services and web APIs, often without the organization's knowledge. As devices continue to proliferate and more applications move to the cloud, businesses are being forced to rethink fundamental assumptions about data access and network security.

Development environments in the cloud allow you to entirely outsource development infrastructure, leaving your team with nothing more than laptops and an internet connection. By using a combination of best-of-breed services such as private GitHub repositories and Snap CI's continuous integration in the cloud, your teams may never need to bother in-house IT for infrastructure again.

Increasing quality and range of choices for inexpensive or free video conferencing is leading to a new way of working for distributed teams. Always-on video connections can help create a sense of **co-location by telepresence**, even when the team is distributed geographically. This is becoming the defacto standard in some of our offshore delivery centers. We are also seeing increased use of screen-sharing tools like ScreenHero for remote pairing. We would caution those looking for a silver bullet to eliminate the need for physical co-location. There is no substitute for the understanding and empathy created by face-to-face communication.

Application configuration can be a source of pain when getting started with a new tool, managing deployments to different environments, or trying to understand why applications behave differently in different places. We are a big fan of minimizing **application configuration**, trying to ensure that applications work sensibly out of the box with the bare minimum of configuration.

Most virtualization technologies provide a way to launch a machine from an image. By creating a **machine image as a build artifact** early in your build pipeline and promoting it through the pipeline as it passes further suites of tests, you can reliably deploy the exact machine that passed the tests into production. This technique eliminates most causes of the snowflake server anti-pattern.

Blue-green deployment is a pattern for performing software upgrades. By setting up the latest version of your application on an identical clone of your production application stack, traffic can be switched, near instantaneously, from the current production stack to the new one as soon as the test suite and the business determine it is appropriate. Though this is an old technique, infrastructure automation and resources in the cloud make it worth reconsidering.

ADOPT
1 Aggregates as documents
2 Automated deployment pipeline
3 Guerrilla testing
4 In-process acceptance testing
5 Mobile testing on mobile networks
6 Performance testing as a first-class citizen
7 Promises for asynchronous programming
8 Windows infrastructure automation

TRIAL
9 Analyzing test runs
10 Blue-green deployment
11 Co-location by telepresence
12 Continuous delivery for mobile devices
13 Database migrations for NoSQL
14 Edge Side Includes for page composition
15 HTML5 storage instead of cookies
16 Logs as data
17 Micro-services
18 Mobile first
19 Perimeterless enterprise
20 Responsive web design
21 Semantic monitoring

ASSESS
22 Capturing client-side JavaScript errors
23 Collaborative analytics and data science
24 Development environments in the cloud
25 Focus on mean time to recovery
26 Machine image as a build artifact
27 Minimizing application configuration

HOLD
28 Exhaustive browser based testing

Techniques

Previously, support for Windows in tools like Chef and Puppet was lacking, leading to large amounts of Powershell scripting to achieve simple infrastructure automation tasks. Achieving the same level of automation for Windows was more challenging than for Unix. In the last 12 months however, both Chef and Puppet support for Windows has improved drastically. That support, combined with the inherent power of Powershell makes **Windows infrastructure automation** extremely viable.

HTML5 storage, also known as local storage or web storage, is a mechanism for storing client side data in modern browsers, including iOS and Android mobile browsers. We recommend using **HTML5 storage instead of cookies** in almost all cases. HTML5 Storage can accommodate up to 5MB of data while cookies are limited to 4KB. Cookie data is transmitted in every request, which slows down your application and potentially exposes data over insecure HTTP connections. In contrast, HTML5 storage data remains securely in the browser. Cookies should be reserved for storing small simple pieces of data like a session ID.

The use of **promises for asynchronous programming** is an old technique that is also known as futures. It is gaining renewed interest in light of the extensive use of JavaScript on both the client and server side. This technique eliminates the use of deeply nested callbacks, flags and pollers and has first-class support from libraries such as jQuery. Teams developing JavaScript codebases of significant complexity should take advantage of this.

Capturing client-side JavaScript errors has helped our delivery teams to 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.

With HTML5 blurring the line between traditional native apps and web apps, we are beginning to experiment with **continuous delivery for mobile devices**. Services such as TestFlight allow you to deploy native apps to real devices multiple times per day. With a wholly or partially HTML5-based application changes can be deployed without submitting a new app to an app store. If your organization has an enterprise app store, you may be able to easily push builds to it. While the techniques for implementing CD to mobile devices are improving, we note that testing practices are lagging behind. To be successful you will need to increase your focus on automated testing to ensure that everything actually works once it gets to the device.

We increasingly see mobile applications that work really well during development and testing, but run into trouble when they are deployed in the real world. **Mobile testing on mobile networks** reveals how your app performs under a variety of conditions. You might test using 3G or LTE or deliberately use a poor WiFi network with overloaded access points. Measure network performance for your target environment, then simulate the conditions using latency and packet-loss inducing tools. In addition, it is sometimes necessary to examine exactly how your device and software are using the network with a tool such as Wireshark.

NoSQL data stores continue to become mainstream, and teams should acknowledge the need for **database migrations for NoSQL**. Especially with an implicit or dynamic schema you are likely to want to reconfigure data over time. There are several approaches such as running an explicit migration when deploying a new build of your application, or using dynamic migrations in code as documents are loaded and processed.

Failing tests reveal bugs in production code. However, **analyzing test runs** for other properties can reveal interesting information. A simple example would be to monitor which tests fail frequently and run them earlier in your build pipeline to get fast feedback. Similarly, tracking other properties such as test execution times and ratios of long-running tests to fast-tests can provide actionable metrics.

In previous radars we recommended arranging automated acceptance tests into longer journeys and, in what we call semantic monitoring, running these tests continuously against a production environment. We still believe that this is an important technique for scenarios the team can anticipate in advance. A variation of this approach, seen especially with startups, is to reduce the number of tests while increasing monitoring and automatic alarms. This shifts the focus from avoiding problems that can be anticipated to **reducing mean time to recovery** for all problems.

While unit and acceptance testing are widely embraced as standard development practices, this trend has not continued into the realm of performance testing. Currently, the common tooling drives testers towards creating throw away code and a click and script mentality. Treating **performance testing as a first-class citizen** enables the creation of better tests that cover more functionality, leading to better tooling to create and run performance tests, resulting in a test suite that is maintainable and can itself be tested.

Tools

In previous radars we have talked about **embedded servlet containers**, and these are now widely adopted on our projects. Tools such as SimpleWeb and Webbit take the simple, embedded approach further and offer raw HTTP server functionality without implementing the Java Servlet specification. At the same time, Tomcat, the most popular Java application server, is increasingly used in embedded setups and Microsoft provides self-hosted servers for the .NET framework, lending further weight to this trend.

D3 continues to gain traction as a library for creating rich visualisations in the browser. Previously, it was somewhat low-level, requiring more work for the creation of commonly used visualisations than less sophisticated, more targeted libraries. Since the last radar, libraries like Rickshaw for charting and Crossfilter for in-browser dataset exploration have helped make D3 even more accessible than before.

We see several JavaScript frameworks embrace **browser-based templating**, moving more layout work to the client. While this approach is useful in many cases, it does introduce operational issues involving caching, performance, and search. We believe these tools should be assessed carefully to ensure suitability for the target deployment environment.

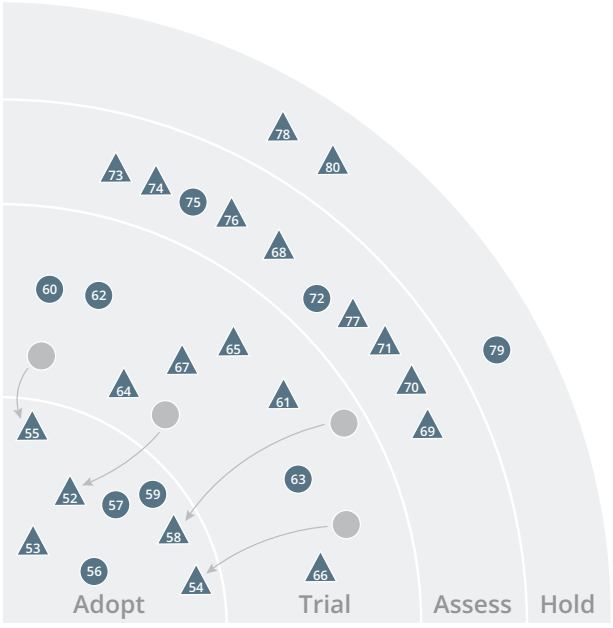
By putting IObservables and IObservers on an equal footing with IEnumerables and IEnumerators, **Rx for .NET** allows developers to use their existing knowledge of LINQ (Language INtegrated Query) operators to query and compose asynchronous operations and event-based code using a common underlying abstraction of observable event streams. Microsoft has also released RxJS to bring the benefits of reactive programming to JavaScript. They open sourced the entire Rx framework, making it useful for Windows rich client applications and single-page JavaScript applications.

Several ThoughtWorks teams called out the usefulness of **Faraday**, a Ruby HTTP client library that provides a common interface over a variety of adapters and integrates nicely with **Rack** middleware.

Package systems for third-party library management continue to gain acceptance and features across all platforms. We called out **NuGet** as a recent entry, and the addition of **Chocolatey NuGe**t exemplifies the advances and capabilities springing up around this essential agile engineering practice.

Windows infrastructure automation should be adopted, however it still remains more difficult than automation on a Unix platform. Tools like Chef and Puppet are increasing their support, but there are also Windows specific solutions being developed like **Octopus**. Octopus allows automated deployment of your ASP.NET applications and Windows services and decreases dependency on PowerShell. It can be used with both NuGet using Octopak and TeamCity to create a full build, package, and deployment pipeline.

Both Puppet and Chef have had to deal with sharing community-contributed modules and manifests for commonly used services and tasks. Both the Puppet Forge and Chef's Cookbook repository have helped, but people ended up copying and pasting these recipes into their own codebases, preventing them from taking advantage of later bugfixes and improvements. **Puppet-librarian and Chef-librarian** attempt to solve this by making it easy to declare your module dependencies, including pulling in known versions of code from these community sites.



ADOPT
52 D3
53 Embedded servlet containers
54 Frank
55 Gradle
56 Graphite
57 Immutable servers
58 NuGet
59 PSake

TRIAL
60 Apache Pig
61 Gatling
62 Jekyll
63 Locust
64 Logstash & Graylog2
65 PhantomJS
66 Puppet-librarian and Chef-librarian
67 TestFlight & HockeyApp

ASSESS
68 Browser-based templating
69 Faraday
70 Hystrix
71 Icon fonts
72 Light Table
73 Octopus
74 Reactive Extensions for .Net
75 Riemann
76 Snowplow Analytics
77 UIAutomator

HOLD
78 Heavyweight test tools
79 Maven
80 TFS

Tools

Managing dependencies in distributed systems can become 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.

Both **TestFlight and HockeyApp** allow you to manage the deployment of mobile applications without an app store, making user testing easier. They offer crash reporting and analytic capabilities to gather data in the field. HockeyApp supports IOS, Android, & Windows Phone, while TestFlight supports iOS and Android. We have used both tools successfully to help deliver mobile applications. This is clearly a fast evolving space.

Frank is an open source library that allows functional tests for iOS written in Cucumber and executed on a remote device. This fills an important niche where acceptance test-driven development was previously cumbersome and awkward.

UIAutomator looks like the most promising tool for testing Android user interfaces by allowing fine-grained control over components during test and facilitating testing on multiple devices.

With the rise of devices with multiple form factors and pixel densities, the issue of presenting high quality icons at all scales has become important. Icon fonts solve this problem by using browser support for WebFonts and SVG instead of scaled images or maintaining different icon sets. As always, when making extensive use of SVG, pay attention to power consumption on mobile devices and performance on older devices.

As the systems we build involve more fine-grained services spread across more machines than ever before, the challenge of how to get information aggregated to allow for easy problem identification and resolution is more pressing than ever. **Logstash** has emerged as an easy way to parse and filter logs at source, and then forward them to a single aggregation point. Although Logstash provides some searching and filtering, **Graylog2** is often used in conjunction to provide for more fully-featured querying and reporting.

We see great promise in **Snowplow Analytics**, an open source web analytics platform that derives intelligent information from regular web analytics, based on open data principles and cloud storage.

We see interest on ThoughtWorks projects around **PhantomJS**, a headless web testing tool that allows functional testing against a realistic target.

Gatling is another newer player in the automated performance testing space. It is similar to Locust and is much lighter weight than the older options such as JMeter and Grinder. Built on Scala, the DSL provides extensive functionality out of the box including easily configured data feeds and response assertions. In cases where customization is needed, it is easy to drop into Scala to provide extensions. The default generation of numerous dynamic views of the data via Highcharts adds to its appeal.

Many organizations that have moved to more agile ways of working continue to use **heavyweight testing tools**. These tools have problems that make them unsuitable for fast moving software delivery. Large complex tools have high learning curves and require specialist skills and training, making it hard for the team themselves to test. Often this results in an unnecessary overhead for every release as other teams get involved. Expensive and limiting software licenses makes this problem even worse. Some heavyweight tools use a “model driven” approach where an attempt is made to accurately model the usage patterns of the application, which leads to costly test script maintenance and development time being lost to “false positives.” We have seen few situations where simple open source solutions cannot give the required level of confidence for much less time, effort and money.

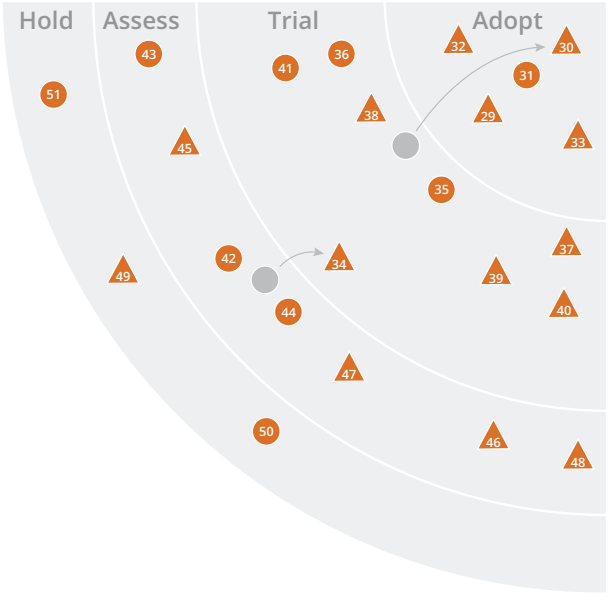
Language-based build tools like **Gradle** and **Rake** continue to offer finer-grained abstractions and more flexibility long term than XML and plug-in based tools like Ant and **Maven**. This allows them to grow gracefully as projects become more complex.

We continue to see teams run into productivity problems attempting to use **TFS** as a version control system. Teams that want to practice frequent code checkins, a core part of continuous integration, have found its heavyweight approach significantly drains productivity. This often leads to teams checking in less frequently, causing more problematic merges. We recommend tools such as Git, Perforce, and Subversion instead.

Platforms

PostgreSQL is expanding to become the NoSQL choice of SQL databases. Version 9.2 includes the ability to store JSON data with full querying capabilities on the content of the JSON document. Other extensions let the user store and query data in the form of key/value pairs. This lets you take advantage of the underlying storage and transactional capabilities of a time-tested database without being tied to a relational data model. This is ideal for those who want both SQL and NoSQL applications but prefer a single reliable infrastructure that they already know how to support.

The amount of data that even a relatively low volume website can generate is huge. Once you add in analytics, business metrics, demographics, user profiles and multiple devices, it can become overwhelming. Many organizations use data warehouses as a repository with data being sucked in from all parts of the organization. The challenge here is that these often turn into "Data Fortresses." Even getting timely business metrics becomes a challenge, let alone running exploratory queries across the entire data set. Technologies like the cloud based **BigQuery** help. The pay-as-you-go model and the ability to do ad hoc queries lets you gain insight without buying specialist hardware and software. A data-driven business should put data in the hands of the decision makers, not hidden behind technological barriers and bureaucracy.



For problems that fit the document database model, **MongoDB** is now the most popular choice. In addition to ease of use and a solid technical implementation, the community and ecosystem contributed to this success. We are aware of problems where teams were tempted by the popularity of MongoDB when a document database was not a good fit or they did not understand the inherent complexity. When used appropriately, however, MongoDB has proven itself on many projects.

Redis has proven a useful tool on multiple ThoughtWorks projects, used as both structured cache and data store distributed across multiple countries.

Hadoop initial architecture was based on the paradigm of scaling data horizontally and metadata vertically. While data storage and processing were handled by the slave nodes reasonably well, the masters that managed metadata were a single point of failure and limiting for web scale usage. **Hadoop 2.0** has significantly re-architected both HDFS and the Map Reduce framework to address these issues. The HDFS namespace can be federated now using multiple name nodes on the same cluster and deployed in a HA mode. MapReduce has been replaced with YARN, which decouples cluster resource management from job state management and eliminates the scale/performance issues with the JobTracker. Most importantly, this change encourages deploying new distributed programming paradigms in addition to MapReduce on Hadoop clusters.

Over the past year we have seen a gradual uptake in the adoption of **Elastic Search** as an open source search platform. It is an extensible, multi-tenanted, and horizontally scalable search solution based on Apache Lucene. It allows complex data structures to be indexed and retrieved through a JSON based REST API. It provides an elegant model of operation with automatic discovery of peers in a cluster, failover, and replication. Elastic Search can be extended with a plugin system that allows adding new functionality and changing existing behavior. The community around this tool is quite vibrant as illustrated by the number of client libraries available in languages like Java, C#, Ruby, and JavaScript.

ADOPT
29 Elastic Search
30 MongoDB
31 Neo4J
32 Redis
33 SMS and USSD as a UI

TRIAL
34 BigQuery
35 Continuous integration in the cloud
36 Couchbase
37 Hadoop 2.0
38 Node.js
39 OpenStack
40 Rackspace Cloud
41 Riak

ASSESS
42 Azure
43 Calatrava
44 Datomic
45 PhoneGap/Apache Cordova
46 PostgreSQL for NoSQL
47 Vumi
48 Zepto.js

HOLD
49 Big enterprise solutions
50 Singleton infrastructure
51 WS-*

Platforms

Node.js is a lightweight web container that is a strong option for development of micro services and as a server to mobile and single-page web applications. Due to the asynchronous nature of node.js, developers are turning to promise libraries to simplify their application code. As the use of promises mature within the node.js community, we expect to see more applications developed for node.js. For those teams that are reluctant to try node.js in production, it is still worthwhile to consider node.js for development tasks like running JavaScript tests outside of the browser or generating static web content from tools like CoffeeScript, SASS, and LESS.

Zepto.js is a lightweight JavaScript library that is largely based on JQuery. The API is identical to JQuery although it does not offer full compatibility with it. With a vastly compressed file size, Zepto is a compelling option when building responsive web applications.

PhoneGap, now renamed as **Apache Cordova**, is a platform that lets you develop cross-platform mobile applications using HTML, CSS and JavaScript. It abstracts away platform specific native code through a set JavaScript APIs that remain consistent across different mobile platforms. Cordova is available for a wide array of platforms including iOS, Android, Blackberry, Windows Phone, and WebOS.

While AWS continues to add more features, **Rackspace Cloud** has become a viable competition in the storage and compute space. Some users may value the more thorough customer support available for Rackspace, as well as the ability to mix in more traditional hosting models. We are not excited about this just because Rackspace is a client of ours and we have had the pleasure developing the platform. We have successfully used Rackspace Cloud with several other clients, and would look forward to it being offered in more geographical locations.

The open source **OpenStack** project is gathering steam, and in recent months is becoming a more viable platform for deploying your own private clouds. Many issues which made OpenStack hard to get up and running have been addressed, and new features are being added all the time. It is clear that the OpenStack consortium and its members like Rackspace, Redhat, and HP are committed to the project as the basis for their own OpenStack-based cloud services.

58% of all phones sold last year globally were feature phones. In many developing countries, this is an even larger majority. If your market requires you to develop for these areas, you need to develop with this constraint in mind. These phones use **SMS and USSD as a user interface**. SMS is a long standing technique for sending messages, and USSD allows you to send SMS like messages in a secure session. You should look at USSD and SMS as another UI and UX platform and treat them as first-class citizens.

Vumi is a scalable open source messaging engine driving conversations through frugal methods on mobile devices. Vumi facilitates SMS, IM and USSD interactions between companies and their clients, health services and their patients, governments and citzens, and more. Vumi integrates with telcos and allows you to build apps on top of it easily. You only have to pay for carrier charges.

The gap between what “enterprise-class” commercial packages provide and what is actually needed is widening. This is especially true for internet facing applications. Innovative solutions that really scale and easily support modern techniques such as continuous delivery are written by practitioners for practitioners. They originate with many internet scale companies and are refined as open source software. **Big enterprise solutions** often obstruct effective delivery due to their accumulated bloat, cumbersome licensing restrictions, and feature sets that are driven by check-lists and imaginary requirements far removed from the realities of most development teams.

Languages & Frameworks

Single-page web application development continues to flourish along with the frameworks supporting data binding, client-side templates, validation, and other capabilities. The **JavaScript MV* frameworks** in active use on ThoughtWorks projects include **AngularJS, Knockout**, and **Ember.js.** Each has advocates and a few detractors. We expect continuing innovative churn in this vibrant space.

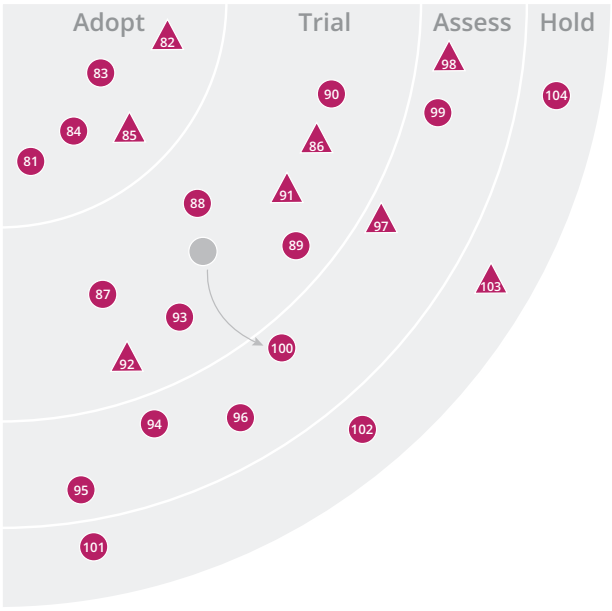
The expansion of single-page and mobile browser-based applications into mainstream use, along with continued growth of node.js for server-side applications, have led to increased adoption of **CoffeeScript** to simplify JavaScript codebases. As a language that compiles into JavaScript code for runtime execution, many concerns have been raised about the difficulty of debugging applications written in CoffeeScript. The introduction of Source Maps in CoffeeScript 1.6.1 is helping producers of development tools address this concern. We expect this will lead to further adoption of the language following the lead of highly visible technology firms such as Dropbox.

Our continued use of node.js on new production applications has re-enforced our need for reliable packaging of JavaScript code and libraries. The **Node Package Manager (npm)** is an important part of the node.js ecosystem and a useful tool for packaging node.js applications. Developers of browser applications with large amounts of JavaScript or CoffeeScript should consider Require.js to help with structuring their code and loading dependencies at run time.

Micro-frameworks are emerging as a way to handle increasing complexity in applications both on client- and server-side. **Sinatra** was one of the first examples of that trend in the server-side space, exposing a lightweight DSL to build fast services that can be easily composed. Similar offerings are available for other languages, including Spark for Java, Flask for Python, Sclatra for Scala, Compojure for Clojure and Nancy for .NET.

One thing that has slowed the evolution of a rich, open source web development ecosystem on the .NET platform has been over-dependence on IIS and the ASP.NET framework. **OWIN** specifies an open HTTP handling interface that decouples web server from application much like Rack has done for the Ruby community. We are excited about OWIN because it opens up the possibility of new .NET web development tools composed of simple, independently-developed modules. Nancy is the perfect example of this. We also hope it will increase the practice of deploying web applications as standalone, self-hosted services on the .NET platform.

The recent release of **Play Framework** 2.1.1 with support for controller dependency injection, asynchronous, non-blocking I/O, a code-reload workflow, database migrations, asset pipelining, and flexible deployment options has made it more attractive to developers. For this reason Play re-appears on the radar as something for teams to seriously consider when building web applications and services on the JVM. A word of caution however, Play embraces a functional programming style which, when working with the Java language, still translates into a plethora of static methods that may be difficult to unit test outside a running server.



ADOPT
81 Clojure
82 CSS frameworks
83 Jasmine paired with Node.js
84 Scala
85 Sinatra

TRIAL
86 CoffeeScript
87 Dropwizard
88 HTML5 for offline applications
89 JavaScript as a platform
90 JavaScript MV* frameworks
91 Play Framework 2
92 Require.js & NPM
93 Scratch, Alice, and Kodu

ASSESS
94 ClojureScript
95 Gremlin
96 Lua
97 Nancy
98 OWIN
99 RubyMotion
100 Twitter Bootstrap

HOLD
101 Backbone.js
102 Component-based frameworks
103 Handwritten CSS
104 Logic in stored procedures

Languages & Frameworks

Along with JavaScript and HTML, CSS is a core technology for creating websites. Unfortunately, the language itself lacks key features, which leads to a high level of duplication and a lack of meaningful abstractions. While CSS3 aims to rectify some of these issues, it will be years before the modules that make up CSS3 will be properly supported in most browsers. Fortunately, there is a solution today using **CSS frameworks** like SASS, SCSS, and LESS. Due to their quality and support, we believe that the days of **handwritten CSS**, for anything apart from trivial work, are over.

CSS frameworks simplify the development of large scale CSS codebases without having to start from scratch each time. Because of the abundance of frameworks, it is important to pick one that enables continued enhancement and maintenance of the codebase, rather than something that just helps you get started quickly. Frameworks based on mix-ins, such as Compass, or with a specific focus, such as Susy, are much better suited in this regard.

Twitter Bootstrap is a popular CSS framework which allows you to quickly produce a good looking website with fluid and responsive layouts. In this edition of the radar, Bootstrap moves back from Trial to Assess based on our experiences using it over time. If you wish to replace or extensively customize the look and feel of your application, Bootstrap can present a challenge due to its deep integration with HTML markup. This doesn't necessarily make it a bad choice but it is worth keeping these limitations in mind when choosing it over available alternatives.

References

Apache Cordova: http://cordova.apache.org/
BigQuery: http://martinfowler.com/articles/bigQueryPOC.html
Client Side Error Logging: http://openmymind.net/2012/4/4/You-Really-Should-Log-Client-Side-Error/
CoffeeScript is not a language worth learning: https://github.com/raganwald/homoiconic/blob/master/2011/12/jargon.md
CoffeeScript Source Maps: http://coffeescript.org/#source-maps
Dropbox dives into CoffeeScript: https://tech.dropbox.com/2012/09/dropbox-dives-into-coffeescript/
DropWizard: http://dropwizard.codahale.com/
Faraday: https://github.com/lostisland/faraday
Graylog2: http://graylog2.org/
Hosted Chef: http://www.opscode.com/hosted-chef/
JavaScript error reporting: http://devblog.pipelinedeals.com/pipelinedeals-dev-blog/2012/2/12/javascript-error-reporting-for-fun-and-profit-1.html
Logstash: http://logstash.net/
Mobile phone Usage: http://www.idc.com/getdoc.jsp?containerId=prUS23982813#.UTTAZzCG2TU
MongoDB: http://www.mongodb.org/
Nancy Framework: http://nancyfx.org/
NPM: https://npmjs.org/
NPM: https://npmjs.org/doc/json.html
Octopus: http://octopusdeploy.com/
OWIN: owin.org
PhantomJS: http://phantomjs.org/
PhoneGap: http://phonegap.com/
Plans: https://github.com/plans
Reactive Extension: https://rx.codeplex.com/
RequireJS: http://requirejs.org/
SnapCI: https://snap-ci.com/
Snowflake servers: http://martinfowler.com/bliki/SnowflakeServer.html
Source Map Revision 3 Proposal: https://docs.google.com/document/d/1U1RGAehQwRypUTovF1KRlpiOFze0b-_2gc6fAH0KY0k/edit
UIAutomator: http://developer.android.com/tools/help/uiautomator/index.html
USSD: http://en.wikipedia.org/wiki/Unstructured_Supplementary_Service_Data
Vumi: http://vumi.org/
Why Everyone Either Hates or Leaves Maven: http://nealford.com/memeagora/2013/01/22/why_everyone_eventually_hates_maven.html

ThoughtWorks is a global IT consultancy

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.