Technology Radar

Prepared by the ThoughtWorks Technology Advisory Board

OCTOBER 2012

thoughtworks.com/radar

ThoughtWorks®

What's new? **Here are the trends highlighted in this edition:**

Mobile—As mobile is rapidly becoming the primary way that people access the internet, this needs to be factored in to new enterprise application strategy, product strategy and implementation – from "mobile first" design all the way through to a new breed of testing tools.

Accessible Analytics—Big Data does not have to equal Big Budgets. A combination of open-source tooling and cloud-based infrastructure provides a more readily accessible analytics and visualization approach.

Simple architectures—Simple continues to gain traction, including both techniques for building and composing applications, as well as infrastructure-based techniques to enable simple deployment, failover and recovery. This theme is a recurring one for us, but we have not yet seen the usage shifts we believe are necessary.

Reproducible environments—Tools supporting the standardization, set-up automation and coordinated management of development, test and production environments for both internally hosted and public cloud environments feature prominently on this edition of the radar.

Data persistence done right—As NoSQL databases are maturing and gaining acceptance, an understanding of the patterns for use (and abuse) becomes imperative.

Introduction

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
Graham Brooks
Hao Xu
Ian Cartwright
James Fischer
James Lewis
Jeff Norris
Mike Mason
Neal Ford
Pramod Sadalage
Ronaldo Ferraz
Sam Newman
Scott Shaw
Srihari Srinivasan
Thiyagu Palanisamy
Wendy Istvanick

Techniques

Adopt

- 1. Health check pages
- 2. Windows infrastructure automation
- 3. Guerrilla user testing
- 4. Work-in-Progress limits
- 5. Automated deployment pipeline
- 6. In process acceptance testing
- 7. Advanced analytics
- 8. Aggregates as documents

Trial

- 9. Polyglot Persistence
- 10. Performance testing as a first-class citizen
- 11. Out-of-container functional testing
- 12. Micro-services
- 13. Infrastructure automation of development workstations
- 14. Agile analytics
- 15. Logs as data
- 16. Responsive web design
- 17. Mobile first
- 18. Declarative provisioning
- 19. Remote usability testing
- 20. Semantic monitoring
- 21. Edge Side Includes for page composition
- 22. Configuration in DNS

Assess

- 23. Deployment and scripting test tools

Hold

- 24. Database based integration
- 25. Feature branching
- 26. Test recorders
- 27. Exhaustive browser-based testing

Tools

Adopt

- 28. Infrastructure as code
- 29. Embedded servlet containers
- 30. Silverback
- 31. AppCode
- 32. Jasmine paired with Node.js
- 33. Immutable servers
- 34. Graphite

Trial

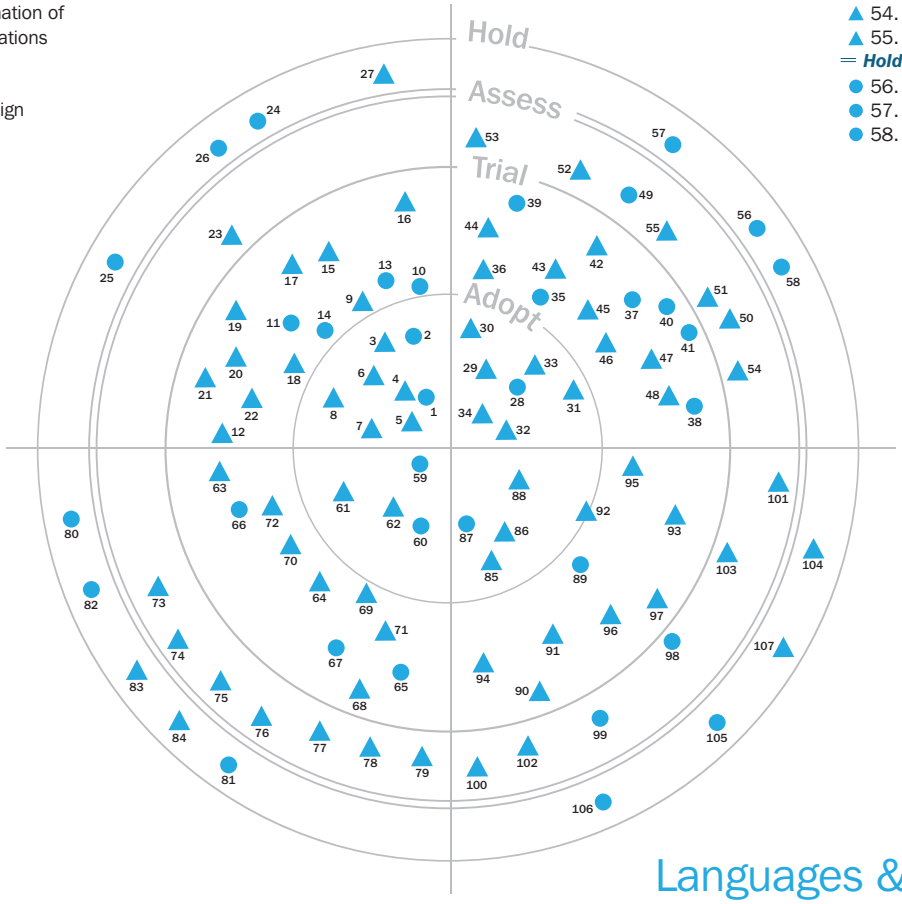
- 35. Vagrant
- 36. Gradle
- 37. PSake
- 38. Frank
- 39. JavaScript micro frameworks
- 40. Jade
- 41. NuGet
- 42. Highcharts
- 43. D3
- 44. Apache Pig
- 45. SaaS performance testing tools
- 46. Dependency Structure Matrices
- 47. Locust
- 48. Rake for Java & .Net

Assess

- 49. Logic-free markup
- 50. Crazy Egg
- 51. Zipkin
- 52. Zucchini
- 53. GemJars
- 54. Light Table
- 55. Riemann

Hold

- 56. Enterprise service bus
- 57. VCS with implicit workflow
- 58. Maven



▲ New or moved
● No change

Platforms

Adopt

- 59. ATOM
- 60. Care about hardware
- 61. Mobile payment systems
- 62. Neo4J

Trial

- 63. Node.js
- 64. Riak
- 65. Domain-specific PaaS
- 66. Linux containers
- 67. Private clouds
- 68. Hybrid clouds
- 69. MongoDB
- 70. Continuous integration in the cloud
- 71. Couchbase
- 72. Single threaded servers with asynchronous I/O

Assess

- 73. Calatrava
- 74. Datomic
- 75. Vert.x
- 76. Azure
- 77. Open source IaaS
- 78. BigQuery
- 79. Windows Phone

Hold

- 80. WS-*
- 81. Java portal servers
- 82. Zero-code packages
- 83. Singleton infrastructure
- 84. Meteor.js

Languages & Frameworks

Adopt

- 85. Clojure
- 86. Scala
- 87. Care about languages
- 88. SASS, SCSS, LESS, and Stylus

Trial

- 89. Domain-Specific Languages
- 90. Scratch, Alice, and Kodu
- 91. Twitter Bootstrap
- 92. Sinatra
- 93. AngularJS and Knockout
- 94. Require.js
- 95. Dropwizard
- 96. Jekyll
- 97. HTML5 for offline applications

Assess

- 98. F#
- 99. ClojureScript
- 100. Lua
- 101. RubyMotion
- 102. Gremlin
- 103. JavaScript as a platform

Hold

- 104. Backbone.js
- 105. Logic in stored procedures
- 106. Google Dart
- 107. Component-based frameworks

Techniques

We are seeing an uptick in adoption of **micro-services** as a technique for distributed system design, both in ThoughtWorks and in the wider community. Frameworks such as Dropwizard and practices like declarative provisioning point to a maturing of the technologies and tools. Avoiding the usual monolithic approach and being sympathetic to the need to replace parts of systems individually has important positive implications for the total cost of ownership of systems. We see this as having greatest impact in the mid-to-long term, specifically with respect to the two-to-five year rewrite cycle.

Breaking up monolithic applications and building systems from micro-services requires a solid strategy to integrate output from disparate systems into a coherent experience for the end-user. Integrating at the presentation layer using **Edge Side Includes (ESI) for page composition** is a practical and elegant solution. This can occur within your environment using a reverse proxy like Varnish or closer to the user in a Content Delivery Network (CDN).

Application deployments often suffer from an excess of environment-specific configuration settings, including the hostnames of dependent services. **Configuration in DNS** is a valuable technique to reduce this complexity by using standard hostnames like 'mail' or 'db' and have DNS resolve to the correct host for that environment. This can be achieved in multiple ways, including split-horizon DNS or configuring search subdomains. Collaboration between development teams and IT operations is essential to achieve this, but that is unfortunately still difficult in some organizations.

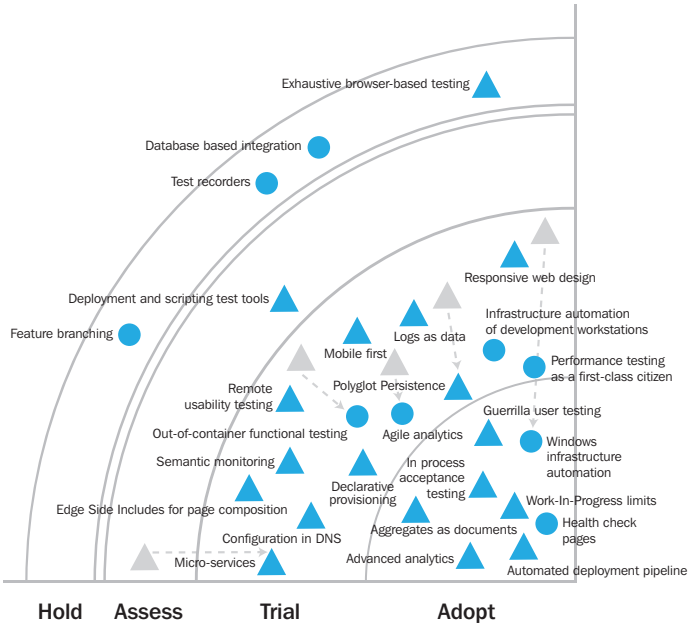
When designing a domain model, the aggregate pattern helps to add structure and modularity. Mapped to a relational database the aggregate is not visible in the table structure. Document databases, like MongoDB, allow you to model **aggregates as documents**. This 1:1 mapping means that the aggregate root should be the object that is loaded from the collection.

The adoption of Continuous Delivery means many teams are creating an **automated deployment pipeline** that carries their code all the way to production. Pipelines allow the visualization of otherwise complex chains of build and deployment activities. Further, they provide the ability to reliably trace build artifacts as they progress through each stage on their path to production. Several vendors are now building CI servers that support the pipeline as a first-class feature and not just a visual element. We recommend teams look closely at these products to avoid wasting time trying to shoehorn a pipeline into a tool without adequate support.

It might sound odd for us to mention this, given how mainstream Agile development has become, but we are noticing teams rediscover and embrace **work-in-progress limits**. Methods such as Kanban limit the amount of in-flight work, forcing better workflow into the team and more visibility into bottlenecks.

Tools such as Pallet offer a compelling approach to environment creation and management through **declarative provisioning**. Usually, this is accomplished by declaring your environment topology - a number of instances, OS, network configuration and applications - using a DSL, and then creating the entire environment automatically via a command-line tool. This approach differs in the decoupling of instance creation and application provision, and in the addition of the ability to declare dependencies between domain-specific application-level services over multiple boxes.

With deployment automation tools maturing, including PowerShell on Windows, scripts are increasingly sophisticated and contain a lot of logic. We recommend **deployment and scripting test tools**, such as Pester for PowerShell and TOFT for Chef and Puppet. It is critical to have good test coverage around the most important aspects of your deployment automation.



Techniques *continued*

We are rapidly heading towards a world where the majority of consumer interactions are from mobile devices. **Mobile first** embraces this trend by designing user interfaces and server interactions that target mobile devices in the first instance. The mobile first strategy contrasts with approaches that assume a highly capable client device connected to a fast and reliable network and then degrade the experience to fit the limitations of the device.

One such technique for achieving this is **responsive web design**. Starting with a basic presentation of content - and typically keeping the essential information constant - the experience is enhanced to suit the features detected on more capable browsers. This commonly takes the form of layout and format changes based on screen size.

Machine learning, semantic analysis, text mining, quantitative analytics, and other **advanced analytics** techniques have steadily matured over the past 15 years. They offer incredible potential for prediction, forecasting, identifying repeatable patterns, and providing insight into unstructured data. Historically, our ability to store and rapidly analyze large amounts of audio, video and image data has been severely limited. This placed constraints on sample size, as well as the time it would take to validate analytical models and put them into production. Now, using a spectrum of new technologies like NoSQL, data harvesters, MapReduce frameworks, and clusters of shared-nothing commodity servers, we have the power necessary to make truly effective use of these techniques. Combined with the massive increase in global data available from sensors, mobile devices and social media and we see this as a field with tremendous opportunity.

Log files generated by web servers, databases, networking infrastructure, and back-end systems are a valuable source of operational and behavioral data for a business. In the past, these files were mostly viewed as a source of diagnostic information in the case of failure, but with lowered cost of storage, and availability of tools such as Splunk for indexing and retrieving millions of events, they can also be a source of customer insights. Treating **logs as data** and storing complete logs instead of just collecting predefined metrics provides a means to answer novel questions that a business could not have previously anticipated.

Bringing users in to a controlled environment for formal testing can be a slow and expensive proposition. Much useful, qualitative feedback can be gathered quickly and cheaply through **guerrilla user testing** - by going out into the world and testing with small samples of the general public. Another alternative is **remote usability testing**, where you can send out everything from wireframes to final applications for testing by people all over the world. Usabila, Loop11 and Treejack all provide tools where you can ask users to carry out specific tasks, and capture everything from the time taken to complete a task, to the user's thoughts and feelings while doing so.

Development teams typically produce tests that specify and validate application behavior, but stop running them once the application goes into production. This is a missed opportunity. **Semantic monitoring** uses your tests to continuously evaluate your application, combining test-execution and real-time monitoring. With micro-services, and similar fine-grained architectural approaches, it is increasingly important to test their interaction at run-time. Incorporating the validation of consumer-driven contracts into a monitoring facility is one way to approach this. While still evolving, we see great promise in the merging of two separate but important verification schemes.

Acceptance tests generally exercise the system from the 'outside', traversing an entire network stack for the security of exercising the complete application. **In-process acceptance testing** challenges the notion that test code and application-under-test must run in different processes in order to achieve these benefits. When using an embedded container, it is easy to set up the system, run the tests over HTTP and to verify the final state without the setup costs associated with deploying to and communicating with a separate container.

We have previously spoken about executing automated tests at the appropriate layer of your application. In this radar, we want to be very specific - we recommend against **exhaustive browser based testing**. Web browser automation tools like Selenium have encouraged widespread automated testing through the browser. While these tests continue to have their place in a test portfolio, most teams find that executing the bulk of tests through the browser creates a slow and fragile test suite.

Tools

Of all the build tools and languages we use across our projects, the one we keep coming back to is Rake. Rake is a beautiful, simple and powerful build language implemented as an internal Domain-Specific Language on Ruby. Ruby's ability to run across several virtual-machine platforms means that Rake is equally available - while leaving open the option to utilize more language-specific tools for some tasks. Finding a similar combination of elegance and flexibility is difficult regardless of your platform, so we recommend trying **Rake for Java and .Net projects**.

Two things have caused fatigue with XML-based build tools like Ant and Maven: too many angry pointy braces and the coarseness of plug-in architectures. While syntax issues can be dealt with through generation, plug-in architectures severely limit the ability for build tools to grow gracefully as projects become more complex. We have come to feel that plug-ins are the wrong level of abstraction, and prefer language-based tools like **Gradle** and Rake instead, because they offer finer-grained abstractions and more flexibility long term.

In a mixed Ruby/Java application, running on the JVM, there are differences in package format and dependency resolution that need to be dealt with. By providing an Ivy compatible proxy that packages RubyGems as JARs and uses Ivy to resolve Gem dependencies, **GemJars** consolidates and simplifies the building of truly polyglot codebases.

Precedents set by cloud providers are now changing expectations within the corporate datacenter. In the cloud, many systems scale automatically, either to provide additional availability or in response to increased demand. Crucial to managing a growing estate, **immutable servers**, or 'phoenix servers', are a sensible approach for enterprises looking at IaaS and PaaS. In contrast, custom-configured 'snowflake servers' increase the load on the operations group and encourage a "works on my machine" mentality. Being able to re-provision machines - hard or virtual - from scratch using tools such as Chef or Puppet can drastically reduce the complexity of managing large server farms. Coupled with software that is designed to withstand failure, this will lead to more scalable and reliable systems.

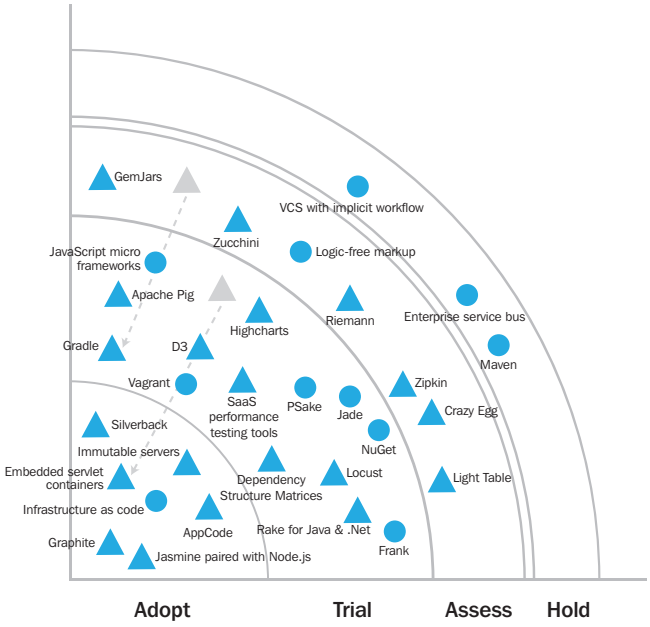
We have long thought of JavaScript as a first class language, and have been keenly following the development of testing tools in that space. The cream of the crop for out-of-browser testing is currently Jasmine. **Jasmine paired with Node.js** is the go-to choice for robust testing of both client- and server-side JavaScript.

When building distributed applications to address web-scale or big data requirements, setting up appropriate monitoring tools becomes a non-trivial exercise. **Zipkin** is a tool that instruments the different components of a service-based system and visualizes the breakdown of logical requests passing through multiple services using a 'firebug-like' view. The raw data can be stored in Hadoop for more advanced reporting and data mining.

Zucchini is a testing framework that provides Cucumber-style BDD testing for iOS apps. It uses CoffeeScript for feature definitions, takes screenshots as tests are run, and we've been very happy with it.

Apple's mobile devices are going strong and native apps are a cornerstone of their success. Writing these native apps has become much more pleasant and productive since JetBrains launched **AppCode**, an IDE for iOS and OS X development that replicates the strengths of their IDEs for other platforms.

Like most good software developers, we choose our tools with care. We are especially keen on interesting departures from the norm, which is why we helped back the **Light Table** Kickstarter project. While still very early in development, the promised interactivity rivals the best of the Smalltalk world, with a modern twist; we are anxious to see what will come of this ambitious project.



Tools *continued*

Hadoop continues to be the most popular framework to develop distributed data-processing applications. Although programming Hadoop applications in Java is not particularly difficult, designing efficient MapReduce pipelines does require a good amount of experience. **Apache Pig** simplifies Hadoop development by offering a high level language, called Pig Latin, and an execution runtime. Pig Latin is procedural and provides a SQL-like interface to work with large datasets. The execution infrastructure compiles Pig Latin into an optimized sequence of MapReduce programs that run on the cluster. Pig Latin is extensible through user-defined functions in different languages such as Ruby, JavaScript, Python and Java.

There are a couple of usability testing tools that match our preferred 'guerrilla' approach. Eye-tracking has long been a useful technique when designing compelling user interfaces, however the equipment and software associated with it is expensive and typically requires the use of specialist firms. **Crazy Egg** is a cheaper, software-only solution that produces heat maps based on mouse movement. This movement has a strong correlation with gaze, and can be used as a reasonable approximation. **Silverback** captures not only the screen during a test, but also records the face and voice of the user. This can be invaluable in sharing rich test experiences with the wider development team.

While many tools exist for displaying graphs for system monitoring purposes, **Graphite** has emerged recently as the clear leader in this space. Capable of charting metrics in real-time, it features a round-robin database that is able to store long periods of historic data, while still providing more recent information at a higher fidelity. Numerous configuration options exist on the dashboard, and the resulting graphs can then be embedded in webpages to increase visibility.

Riemann is an open source server that aggregates and relays events in real time. Written in Clojure, and based on Netty, it is capable of handling thousands of concurrent connections per node. Riemann uses a simple Protobuf protocol for events, which allows it to aggregate everything from CPU and memory use to orders placed to error rates. It forwards to systems like Graphite, triggers email alerts, and provides a dashboard for monitoring these metrics. Riemann is an important part of the movement towards handling data as generic streams of events in real-time, as opposed to using specialized systems for different types of data.

Increasingly performant JavaScript engines, combined with widespread support for embedded SVG documents in HTML, has lead to pure JavaScript-based client-side graphing and visualization solutions gaining a lot of traction. **Highcharts** is one of the best ones we have come across, with out-of-the-box support for multiple highly-configurable interactive chart types, and the ability to easily render large data sets.

D3 is a JavaScript library for binding datasets into the DOM, and then declaratively transforming the document to create rich visualizations - ranging from graphs to heatmaps. With support for HTML, CSS and SVG, and an extensible plug-in model, we like the fact that this library allows us to deliver information in more intuitive ways.

We strongly favor code-base visualization techniques. In particular, **Dependency Structure Matrices** (DSM) have proven to be extremely useful, especially in support of an evolutionary architecture and emergent design. Tools support for DSM is widespread.

We have talked much already 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. We are pleased to see a corresponding reduction in the complexity of test code that takes advantage of this.

We are strong believers in in-line automated performance testing, although open source tools in this space have been somewhat limited to date. **Locust** is a firm favorite that provides the ability to write tests in Python, with good support for running multiple injectors, basic statistics generation, and a useful web dashboard. Its approach to web load testing focuses more on the simulation of users than just generating hits per second. We would typically recommend Locust over and above older tools such as JMeter or Grinder.

Rather than wrestling with licenses and setting up clusters of machines for performance testing, we're seeing a rise in **SaaS performance testing tools** such as Blitz.io and Tealeaf. These services make it easy to run performance tests with a huge number of geographically diverse clients, without investing heavily in infrastructure.

Platforms

Hybrid clouds combine the best features of public clouds and private data centers. They allow applications to run in a private data center during normal periods, and then use rented space in a public cloud for overflow capacity during peak traffic periods. There are now a number of infrastructure solutions that allow automatic and consistent deployment across a hybrid cloud, such as Palette and RightScale. With robust offerings from Amazon, Rackspace and others, we are moving hybrid clouds to "Trial" on this edition of the radar.

Selecting the right cloud provider from an almost bewildering array of options continues to be difficult. One strategy is to adopt an **open source IaaS** platform such as OpenStack or CloudStack. This allows you to run a private cloud that is consistent with a public cloud, and to migrate from one cloud provider to another should the need arise. Going one step further, Apache's Deltacloud abstracts away from specific provider APIs to give a consistent experience across cloud platforms.

Google's **BigQuery** brings data analytics to the cloud. Rather than loading data into an expensive data-warehouse with predefined indexes, BigQuery allows you to upload and investigate a data set through ad-hoc SQL-like queries. This is a great way to create a cheap proof-of-concept or even a complete application, as processing of hundreds of gigabytes of data by thousands of servers happens in seconds.

Microsoft's **Azure** cloud platform continues to play catch-up with more mature clouds such as AWS, but we've been impressed with how Microsoft has responded to market demands. As with most Microsoft solutions it continues to be a contender and worth evaluating.

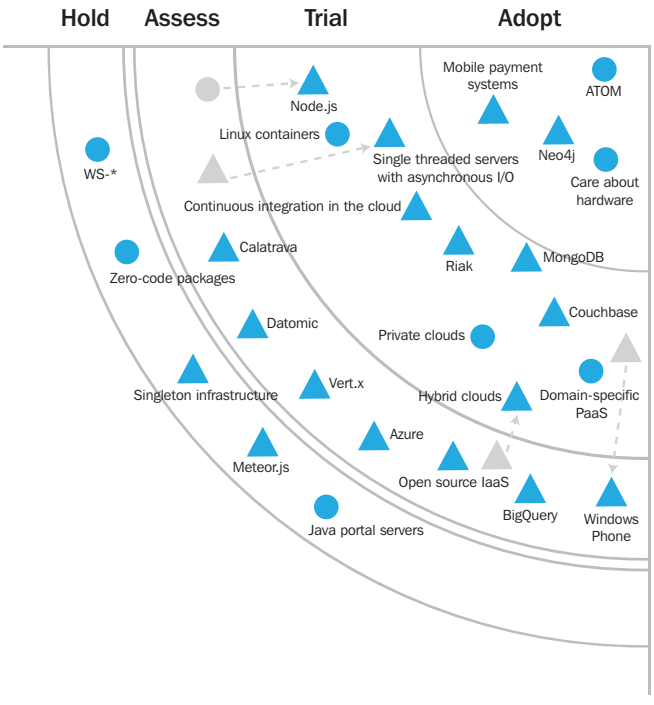
Continuous integration in the cloud is one of those obvious-in-hindsight infrastructure offerings that supports agile development. With no local software and minimal configuration, it just works. With mature offerings now in place, serious developers are left with no excuse for avoiding this important practice.

Despite apparent resistance in the Global North, **mobile payment systems** such as Kenya's M-Pesa are providing secure cashless monetary transactions. With the service rolling out across Africa, the system opens up the market for the millions of people with mobile phones but lacking access to traditional banking outlets. Providers such as Square are slowly improving the situation, but the North continues to lag.

For problems that fit the document databases model, **MongoDB** provides easy programmability, a query interface, high availability with automated failover, and automated sharding capabilities. It allows for a smooth transition to NoSQL data stores from the RDBMS model, with the inclusion of familiar concepts, such as the ability to define indexes.

Graph databases store information as arbitrarily interconnected nodes linked by named relations, rather than as tables and joins. Schema-less and highly extensible, they are an excellent choice for modelling semi-structured data in complex domains. **Neo4j** is the front-runner in the space - both its REST API and its Cypher query language support simple and fast storage and traversal of graphs.

Riak is a distributed key-value store that is schema-less and data-type agnostic. It can be put to good use in write-heavy projects to store data such as sessions, shopping carts and streaming logs - whilst it retains the ability to perform complex queries in a full-text search. The distributed cluster can self-recover without a single master, has tuneable consistency and availability settings and can do collision detection and resolution if needed - all of which can be particularly helpful in high availability environments.



Platforms *continued*

A fundamental rethinking of how databases work, **Datomic** is an immutable database server with fascinating transactional and deployment characteristics. One of the common headaches on agile projects is managing database migrations, especially restoring previous states. Datomic makes the need for migrations go away - every version of the data (and schema) is preserved by the database. While still evolving, we appreciate Datomic's boldness of vision.

Couchbase is a persistent cache with auto-sharding features, master-less clusters and replicated data to avoid cache-misses. Because it supports the Memcached protocol, it allows drop-in replacement for Memcached based systems.

Representing yet another evolution away from traditional, free-standing application containers, **Vert.x** is an application framework that bridges synchronous and asynchronous programming styles. This gives the programmer the option to trade off scalability and performance for simplicity. Unlike Node.js, Vert.x is a library that can be called from a variety of languages supported on the JVM, including Java, Ruby and JavaScript.

We have previously been skeptical of claims of reusable code working across platforms. Our experience with many tools in the market has been mixed and we advise caution to our clients who are looking at these types of solutions. Taking an approach that carefully navigates these dangerous waters, we feel **Calatrava** is worth evaluating for mobile application development. The framework neatly follows the separation of business and presentation logic, maximising reuse where there is commonality, and providing native access where speed or device-specific idioms are to be followed.

Meteor.js is a client- and server-side JavaScript application framework, run inside a web browser, or in a Node.js container, and backed by MongoDB for persistence. It uses "Smart Packages" - little bundles of code that can run in the browser or as part of a cloud service. It allows hot code deploys and live in-browser updates. We think the idea is great, even if the framework is not yet ready for primetime.

Despite a promising start to **Windows Phone**, a well thought-out user interface, and probably the best development experience of any mobile platform, we have seen several stumbles in the execution of the platform strategy by Microsoft and its partners. This makes us less optimistic about the future of the platform than we were in the last radar.

Sometimes, architectural decisions lead you to incorporate infrastructure items that you can only afford one of, such as mainframes or search appliances. This is a terrible idea. It severely restricts testing and deployment flexibility. We strongly favor infrastructure you can easily set up and tear down. **Singleton infrastructure** belongs to misguided vendor-driven architectures of the past.

Languages & Frameworks

JavaScript is moving outside of the browser, emerging as an important technology for cross-platform development. It is front-and-center in the approach to code reuse taken by Node.js, Meteor.js and mobile frameworks like Calatrava. Along with the recent proliferation of other languages that compile to JavaScript, this makes us wonder if we should start to consider **JavaScript as a platform** and not just a language.

As adoption continues to expand, so does the size of many JavaScript codebases. To improve modularity of code and help manage this, we are seeing teams embrace libraries such as **Require.js**. Using the Asynchronous Module Definition (AMD) format, code is split into modules, easing development and maintenance, and an optimization tool then combines and minifies scripts for production deployment.

With JavaScript development on the rise, there is a growing need for reusable, extensible UI tooling. **Twitter Bootstrap** builds on the best offerings in the space, to provide a powerful set of patterns and components that help developers create responsive and adaptive applications with pleasant aesthetics.

We think it is essential to inspire the next generation of technologists. **Scratch, Alice, and Kodu** are programming languages that rely on visual environments and building blocks as teaching devices. They offer exciting possibilities for educational programs and organizations intending to foster programming knowledge in environments beyond academia.

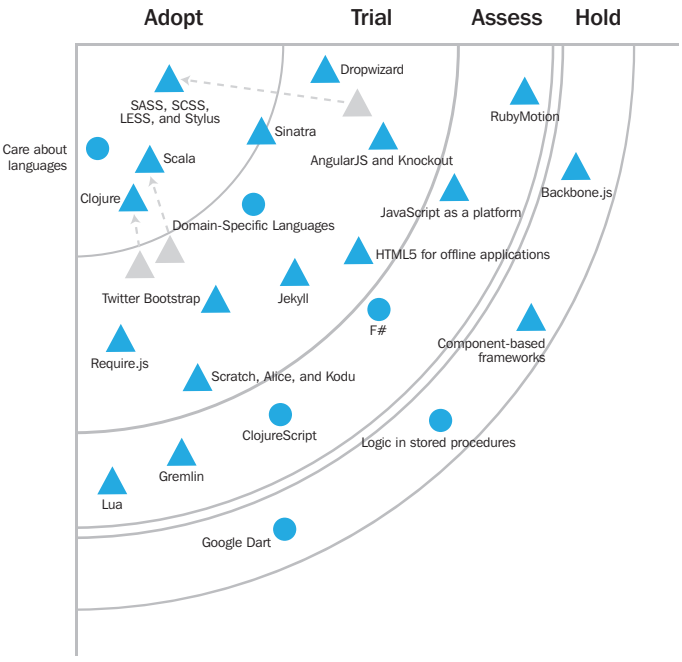
An unlikely contender in the programming languages space, **Lua** has seen massive adoption across a variety of industries. It is used as a scripting platform in game development and music composition; embedded in point-of-sale appliances and network devices; and in extending NoSQL databases with safe execution semantics. We expect further growth in time to come.

Micro-frameworks are emerging as a way to handle increasing complexity in applications both on client- and server-side. **Sinatra** was one of the early precursors of that trend in server-side space, exposing a lightweight DSL to build fast services that can be easily composed. Flask, Scalatra and Compojure are similar offerings for Python, Scala and Clojure respectively.

Dropwizard is an opinionated combination of several lightweight Java tools and frameworks, many of which would merit mention in their own right. The package embodies many of our favorite techniques, including an embedded HTTP server, support for RESTful endpoints, built-in operational metrics and health-checks, and straightforward deployments. Dropwizard makes it easy to do the right thing, allowing you to concentrate on the essential complexity of a problem rather than the plumbing.

Gremlin is an imperative graph traversal language supported by multiple graph databases. Its concise constructs can be used in place of the native language of the database, leading to faster development times and, in some cases, faster execution. We recommend its use as a good alternative in simple scenarios.

Jekyll represents the “microization” of frameworks in the web publishing space. While the focus is maintained on doing one thing - sites that feature blogs - as transparently as possible, it also shows the path to a more lightweight future. One example of this that we like is that it is now trivially easy to publish useful documentation for your software project.



Languages & Frameworks *continued*

Introducing a Ruby compiler and toolchain for developing iOS applications, **RubyMotion** has unsurprisingly caused quite a stir in the ThoughtWorks development community. There continues to be a need to understand the underlying iOS APIs and some Objective-C when building applications, but there are clear benefits for those who find working with the Ruby language and tools more comfortable.

There is a tendency to equate the need for offline functionality with the need to build an app. Despite the slow standardization process, most HTML5 features have now been implemented across all major browsers. Its local storage capabilities, comprehensively supported across mobile and tablet browsers - makes **HTML5 for offline applications** a very suitable option.

We are seeing a common pattern of creating single-page web applications. Rather than requiring full page refresh, these request smaller sets of data from the server, and change the displayed content of their page through modifying the DOM. To make this more manageable, JavaScript MV* frameworks have been developed that support data binding, client-side templates, and validation. While lightweight applications may not need a framework, for more complex scenarios, **AngularJS and Knockout** should be considered as the current front-runners in this field.

Backbone.js is a great example of an abstraction pushed too far. While we initially liked the ease of wire-up, in practice it suffers from the same issues as all such data-bound frameworks from WebForms to client/server tools. We find that it blurs the framework and model too much, forcing either bad architectural decisions or elaborate framework hackery in order to preserve sanity.

As the industry shifted from desktop GUI development to the web, it seemed natural to port the most successful patterns and designs to the new paradigm. After 15 years of trying, we feel that there are still no **component-based frameworks** that have successfully achieved this. We recommend not attempting to make web development into something that it fundamentally is not. It is time to accept the page and request-based nature of the web, and focus on the frameworks that support - rather than work against - these concepts.

References

Alice http://alice.org
Apache Pig http://pig.apache.org
Calatrava http://calatrava.github.com
D3 http://d3js.org
Dropwizard http://dropwizard.codahale.com
Light Table http://kodowa.com
Locust http://locust.io
Lua http://lua.org
Riemann http://aphyr.github.com/riemann
Scratch http://scratch.mit.edu
Silverback http://silverbackapp.com
Zucchini http://zucchiniframework.org

ThoughtWorks is a global IT consultancy

ThoughtWorks – The custom software experts. A company wholly devoted to the art and science of custom software. We make it, and we make our clients better at it. Our bottom line is to design and deliver software fast and predictably. Doing enterprise-scale software is tough, but the returns to those organizations that can deliver – on target – are tremendous.

ThoughtWorks' products division offers tools to manage the entire Agile development lifecycle through its Adaptive ALM solution™, comprised of Mingle®, Go™ and Twist®. ThoughtWorks employs 2,100 professionals to serve clients from offices in Australia, Brazil, Canada, China, Germany, India, Singapore, South Africa, Uganda, the United Kingdom and the United States.

We lead the industry in rapid, reliable and efficient custom software development.
When you need an expert partner to help you get ahead and stay ahead of the competition, get in touch. http://www.thoughtworks.com/contact-us