Technology Radar

http://www.thoughtworks.com/radar

Prepared by the ThoughtWorks
Technology Advisory Board

July 2011

ThoughtWorks®

What's new?

Since the last publication of the Technology Radar, these technology trends are most prominent:

- Tools for effectively delivering and testing mobile web
- Simple techniques for testing and obtaining performance
- Several new approaches to business intelligence
- Continued emphasis on continuous delivery and web based architectures

Introduction

The ThoughtWorks Technology Advisory Board is a group of senior technology leaders within ThoughtWorks. They produce the ThoughtWorks Technology Radar to help decision makers understand emerging technologies and trends that affect the market today. This group meets regularly to discuss the global technology strategy for ThoughtWorks and the technology trends that significantly impact our industry.

The Technology Radar captures the output of these discussions in a format that provides value to a wide range of stakeholders, from CIOs to enterprise developers. The content provided in this document is kept at a summary level, leaving it up to the reader to pursue more detail when needed. The goal of the radar is conciseness, so that its target audience understands it quickly. The radar is graphical in nature, grouping items into techniques, tools, languages and platforms. Some radar items could appear in multiple quadrants, but we mapped them to the quadrant 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 moved since the last radar are represented as triangles, while items that have not moved since the last radar are represented as circles. As we look at each quadrant in detail, we show the movement that each item has taken since the last publication of the radar. Items that have not moved in two publications of the radar fade and are no longer displayed unless something significant happens.

Contributors

The ThoughtWorks Technology Advisory Board is comprised of

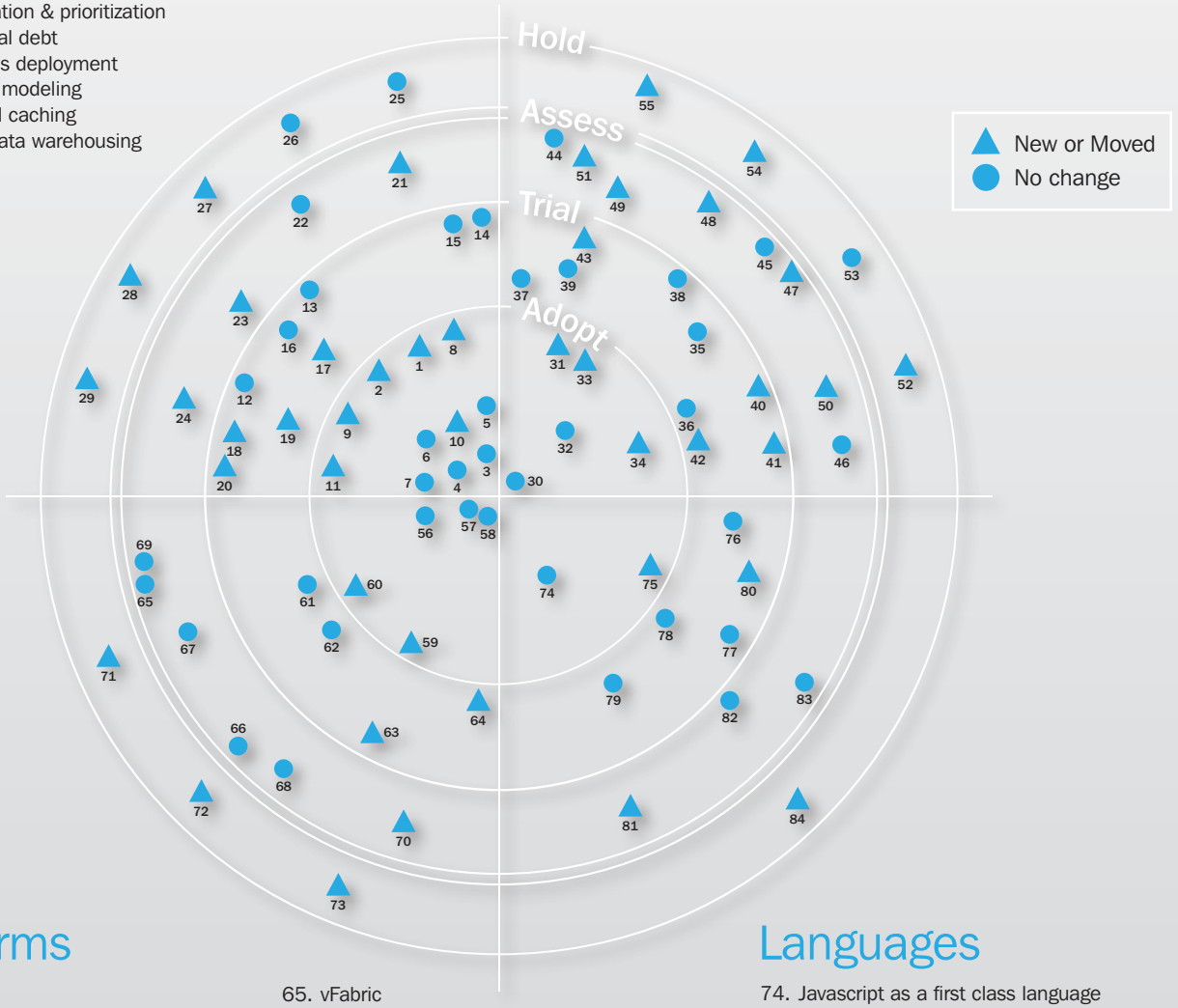
Rebecca Parsons (CTO)
Martin Fowler (Chief Scientist)
Nick Hines (CTO Innovation)
Evan Bottcher
Graham Brooks
Ian Cartwright
Erik Doernenburg
Neal Ford
Ajey Gore
Wendy Istvanick
Mike Mason
David Rice
Pramod Sadalage
Samir Seth
Scott Shaw
Hao Xu
Jeff Norris

Techniques

- ▲1. Progressive enhancement
- ▲2. Automate database deployment
- 3. Platform roadmaps
- 4. Evolutionary database
- 5. Emergent design
- 6. Visualization and metrics
- 7. Coding architects
- ▲8. Evolutionary architecture
- ▲9. DevOps
- ▲10. Simple performance trending
- ▲11. Continuous delivery
- 12. Concurrency abstractions and patterns
- 13. Acceptance test of journeys
- 14. Categorization & prioritization of technical debt
- 15. Continuous deployment
- 16. Capability modeling
- ▲17. Thoughtful caching
- ▲18. Iterative data warehousing
- ▲19. Build your own radar
- ▲20. Event API's
- ▲21. Event driven business intelligence
- 22. Smart systems
- ▲23. Event sourcing
- ▲24. Decision driven BI
- 25. Scrum certification
- 26. Database based integration
- ▲27. Procedure oriented integration
- ▲28. Feature branching
- ▲29. Manual infrastructure management

Tools

- 30. Subversion
- ▲31. Git
- 32. Infrastructure as code
- ▲33. Github
- ▲34. Caching reverse proxies
- 35. Splunk
- 36. Mercurial
- 37. Message buses without smarts
- 38. NoSQL
- 39. Next gen test tools
- ▲40. New Relic beyond Rails
- ▲41. TLB
- ▲42. Powershell
- ▲43. Selenium 2 testing of mobile websites
- 44. Deltacloud
- 45. Vagrant
- 46. API management services
- ▲47. jQuery Mobile
- ▲48. Backbone.js
- ▲49. Sonar
- ▲50. Open source bI tools
- ▲51. Gradle
- ▲52. Cross platform mobile toolkits
- 53. ESB
- ▲54. VCS with "implicit workflow"
- ▲55. Code in configuration



Platforms

- 56. JRuby
- 57. ATOM
- 58. KVM
- ▲59. AWS
- ▲60. Mobile web
- 61. Heroku
- 62. Tablet (formerly iPad)
- ▲63. Offline mobile webapps (just html5)
- ▲64. Ubiquitous computing
- 65. vFabric
- 66. OpenStack
- 67. Node.js
- 68. OAuth
- 69. GPGPU
- ▲70. Cloud Foundry
- ▲71. WS-*
- ▲72. GWT
- ▲73. Java portal servers

Languages

- 74. Javascript as a first class language
- ▲75. HTML 5
- 76. SASS, SCSS, and LESS
- 77. HAML
- 78. Domain-specific languages
- 79. Scala
- ▲80. Coffeescript
- ▲81. Clojure
- 82. F#
- 83. Future of Java
- ▲84. Logic in stored procedures

Techniques

If you are wondering "What comes after agile?," you should look towards **continuous delivery**. While your development processes may be fully optimized, it still might take your organization weeks or months to get a single change into production. Continuous delivery focuses on maximizing automation including **infrastructure as code**, environment management and deployment automation to ensure your system is always ready for production. It is about tightening your feedback loops and not putting off anything until the end. Continuous delivery is not the same as **continuous deployment**, which means deploying every change to production. Continuous delivery is not a cowboy show. It puts you in charge of your production environment. The business can pick and choose what and when to deploy. If you think you've nailed agile development, but aren't considering how to achieve continuous delivery, you really haven't even started.

Improving the interactions and relationship between development and IT operations gives us more effective delivery and production systems that are more stable and maintainable. Creating a **DevOps** culture requires attention to team organization, work practices, reporting lines, and incentives - leading to joint responsibility for faster and safer delivery. We recommend adopting DevOps because we cannot see any situation where attention in this area will not have a positive benefit.

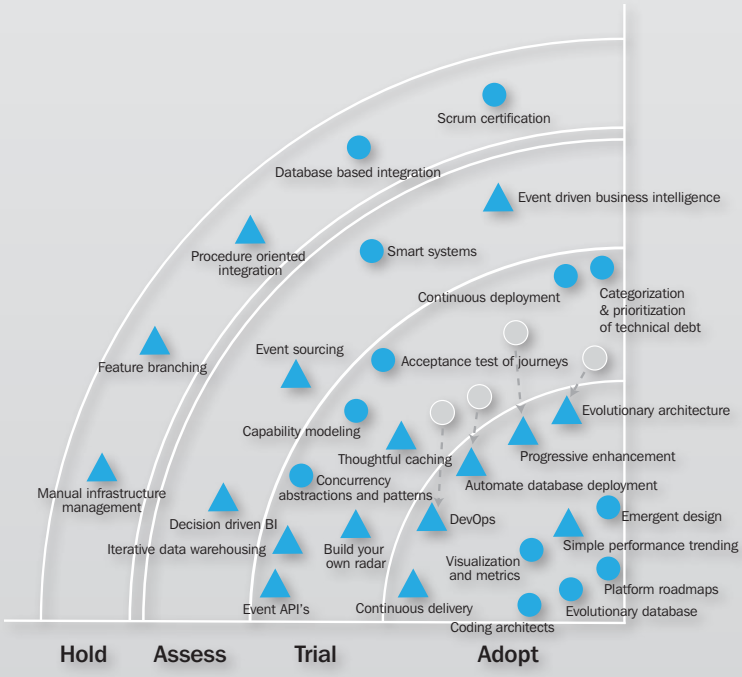
In contrast to traditional up-front, heavy-weight enterprise architectural designs, we recommend adopting **evolutionary architecture**. It provides the benefits of enterprise architecture without the problems caused by trying to accurately predict the future. Instead of guessing how components will be re-used, evolutionary architecture supports adaptability, using proper abstractions, database migrations, test suites, continuous integration and refactoring to harvest re-use as it occurs within a system. The driving technical requirements for a system should be identified early to ensure they are properly handled in subsequent designs and implementations. We advocate delaying decisions to the latest responsible moment, which might in fact be up-front for some decisions.

RESTful APIs have become standard in our industry. A good REST API provides a simple, lightweight means of building customizations and integrations. However, many of the quick, high value integrations we'd like to build require knowing when something happened. Consider building an **event API**, which, when used in conjunction with a REST API, facilitates simple workflow, notification, and synchronization integrations. These integrations often require no more than 20 or 30 lines of code. Often event APIs take the form of a "web hook" or callback mechanism, but don't be afraid of using a poll-based Atom style either. An Atom event API scales cheaply and gives the client the power to guarantee delivery.

One of the goals of SOA has been to decouple services by exchanging human-readable business documents instead of programming parameters. However, in implementing SOA, many businesses have simply used web services to expose the underlying programming models of back-end systems. **Procedure oriented integration** is nothing more than remote procedure calls implemented via a different protocol. The consequences of this are additional layers of complexity with no improvement in business flexibility. To avoid this, implementers of SOA should first understand the business meaning of their services, then implement human-readable contracts that are independent of legacy system implementation.

All too often caching is an afterthought used to address performance problems with a blanket approach and common cache lifetime. This leads to issues and workarounds. The "time value" of information is inherently linked to the business purpose and hence needs to be captured at the same time as other requirements. We believe **thoughtful caching** should be addressed early in the project and not just treated as a last minute performance fix.

Starting performance tests late in a project is risky and costly. Very simple performance tests that exercise key parts of the system, run on a regular basis, are good enough to track trends, so we can react if we see a change in performance. Run these tests with your build or as an overnight job and graph the results to create **simple performance trending**. Complex performance tests in a truly representative environment are still useful, but don't wait for them to start understanding how the performance of your code is changing.



Techniques *continued*

If the rate at which business is changing is an indicator of change in requirements, then the days of doing upfront database design are gone. Instead, projects should follow **evolutionary database** techniques and continue to change their database schemas as new requirements are implemented over the course of the project. Deployment of database changes should also be automated so that the application release that relies on those changes does not have to wait for manual deployment of the database changes. **Automated database deployment** ensures that application and database changes can be deployed automatically. Evolutionary database and automated database deployments ensure highly productive teams a path to continuous delivery.

Despite advances in automation, many people fall back on **manual infrastructure management**. We often see problems caused by manual configuration of firewalls and load balancers, and especially by DBAs cutting and pasting SQL scripts to run against production databases. All of these activities, if not fully automated, should at least be scripted and repeatable across environments.

Disappointingly, we continue to see development teams embrace the practice of **feature branching** to isolate work and defer integration. Feature branches commonly cause significant pain and unpredictability during late merges, but more importantly prevent the continual design improvement necessary to maintain high quality software. We recommend continuous integration and branch by abstraction as an alternative to feature branching.

Recent use of **progressive enhancement** with mobile applications has been very effective and demonstrates the universal nature of this web design strategy. We encourage people to adopt this strategy to keep their code clean and give each user the optimal experience for their device.

Event sourcing is an approach to thinking about persistent data where the primary record is a log of all events that make updates. A traditional representation of database state can be entirely recreated by reprocessing this event log. Event sourcing's benefits include strong auditing, creation of historic state, and replaying of events for debugging and analysis. Event sourcing has been around for a while, but we think it is used much less than it should be.

Traditional approaches to implementing data warehouses and business intelligence work from the bottom up in horizontal tiers, assembling and cleansing data sources from across the enterprise then aggregating them into a comprehensive data mart before reports can be generated. Some people are now employing an alternative approach that starts with the real outcome--a business decision--and pulls work items through the process as needed to support that decision. **Decision driven business intelligence** allows a more incremental approach to BI and facilitates rapid feedback to the decision makers who are the ultimate consumers of business intelligence.

Like iterative software development, there is lot of value to be gained by delivering data warehousing projects using iterative techniques. **Iterative data warehousing** techniques allow the end users of the data warehouse to determine what reports they want and the ETL developers and data modelers to deliver those features without wasting time with data modeling and ETL jobs that do not provide immediate value to the business.

Data visualizations have been effective in business and IT decision making. Organizations are making effective use of real time data through visualizations. These visualizations include point in time data as well as trends plotted over time. We are seeing increased adoption of these techniques in optimizing operations and software development.

Building your own technology radar helps you decide, normalize, and publicize consensus technology views for all interested parties. ThoughtWorks produces a technology radar for clients and friends, telling the world our opinions about upcoming technology trends. You should do this for your own company as well. Too many decisions in large companies happen in a vacuum, with no input from the technologists who have to live with them every day.

Tools

The DevOps movement continues to grow, with developers and operations staff working closely together to solve the "software last mile" problem. **Infrastructure as code** is a technique for treating infrastructure configuration in the same way as code; checking it into source control, then using it to push changes out to the data center. In addition to web server, application server and application configuration, we are seeing network configuration treated in the same way. Network switch, firewall and load balancer configuration can be infrastructure as code, and even changed at runtime.

Measuring software internal quality is still a mystery, even though many source code metrics have been around for years. The problem with those metrics is they usually only capture one aspect of quality. We must consult many metrics to come to a conclusion about the overall quality of our code. **Sonar** is an integrated tool for checking, tracking and visualizing those metrics. It not only combines metrics together, but also mixes them with historical measures, giving us a better insight into the internal quality of the codebase.

Many organizations try to minimize change in production IT environments. This frequently leads to behavioral anti-patterns. One example of this is over use of **code in configuration** to affect the behavior of production systems. Changes that really belong in code end up in configuration files which don't necessarily pass through the same levels of testing as the application. Streamlining the path to production and focusing on quality simplifies rather than complicate things.

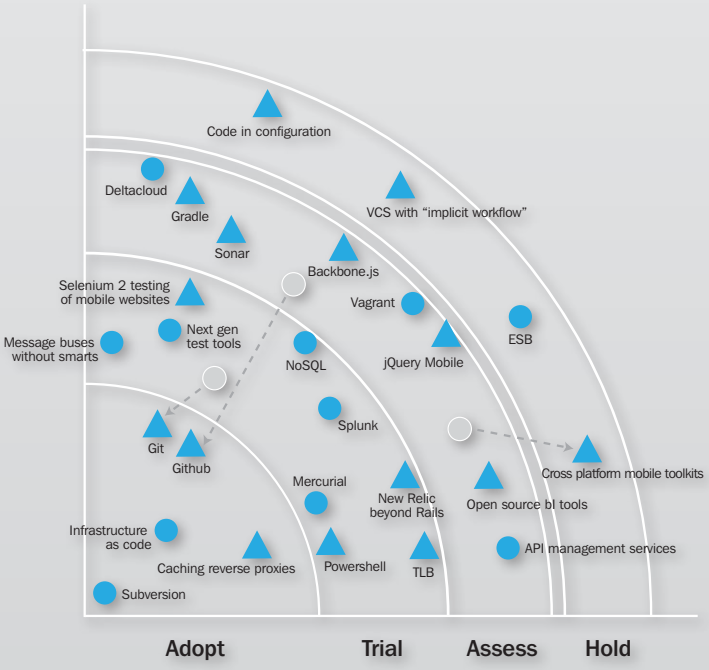
If your test suites are growing slower and you have already verified that it is not a serious problem with your application, first make your tests faster, then look at parallelization. The **Test Load Balancer (TLB)** project is a big development in the world of parallel test execution. It removes the inefficiencies of manual work distribution using smart algorithms and historical test execution data to optimize workload distribution and minimize elapsed time. Further, it orders the tests in intelligent ways like executing the test that failed in the previous execution first to get quicker feedback. Parallel execution can occur across a grid of machines or across multiple processes on a single machine. JUnit, RSpec, Test::Unit, Twist and Cucumber are currently supported and NUnit is under development.

Application designs that incorporate **caching reverse proxies** as first class design elements are simpler and more resilient to infrastructure failures. Placing a caching reverse proxy between an application and a web service it consumes reduces the risk of service failures affecting the application while improving overall system performance.

Even though JavaScript increasingly plays a more important role in today's world of software development, it is still troublesome to organize in a clean structure. **Backbone.js** is a library which provides an MVC (model view controller) framework for JavaScript heavy applications. It allows developers to write JavaScript code in a more manageable and testable way.

Starting from a challenge posed to the Linux community to stop using commercial version control, **Git** has proved itself. Git embodies a well architected, high performance implementation of distributed version control. Git is powerful, so it should be used with respect, but that power enables agile engineering workflows that simply cannot exist with other tools. Git's popularity is supported by the existence of **GitHub**. GitHub combines public and private Git repositories, social networking, and a host of other innovative tools and approaches.

Some tools seek to enable and facilitate different ways of working. Unfortunately other tools are created using a different premise, one of low trust in users and the need to enforce a predefined process. ClearCase and TFS do this. This makes **version control systems with "implicit workflow"** unsuitable tools for modern agile software development. Project methodologies and the best ways



Tools *continued*

of working on a project need to emerge. Tools that enforce high ceremony around things like check in just get in the way and kill productivity.

Thoughtworks has used **jQuery Mobile** on two projects with mobile websites and had mixed experiences. One project found the library very useful for dealing with device differences and graceful degradation on older browsers. On this project we were working in a way that fit with the jQuery Mobile approach. Our other project found the tool less useful and felt to some extent it was trying to force them to work a particular way that did not fit their application well. For these reasons we have decided to leave this tool in assess. If you are doing mobile web it is definitely worth spiking but it may not fit every application.

When building mobile web applications we can now use **Selenium 2 mobile tests** to run the same acceptance tests on iOS, Android and Blackberry. This works on emulators, simulators and physical devices. We have successfully used this approach on production software for all 3 platforms. While the Blackberry driver is still in beta, we found it stable enough for use. The key challenge is the different ways to install the driver and start the browser, but this only needs to be solved once. We suggest that companies doing mobile web for these devices try this approach. We see no reason why this approach cannot be extended to Windows Phone in the future.

With very few exceptions, tools that claimed to create seamless user experiences across Windows, Linux and OSX did not deliver. We ended up with compromised experiences on one or more of the operating systems. Mobile adds complexity to this problem with different hardware form factors and conventions for user interactions. We have made several attempts to use **cross platform mobile toolkits** on our projects with varying degrees of success. We saw issues like having to create a project for each platform or invoking specific native UI widgets to get things working. For these reasons we have put cross platform mobile toolkits in hold. While this may change in the future, we remain skeptical especially given past experiences on hardware that was far more homogeneous.

NoSQL technologies are maturing daily, allowing for innovative solutions as businesses need to scale massively or ask intelligent questions of existing data. Technologies like MongoDB, Riak, Neo4J, Cassandra and many others are helping power the NoSQL space.
Open source BI tools such as Pentaho, JasperSoft, CloverETL, Talend, BIRT and SpagoBI are matching features with the proprietary tools and allowing for easy entry into the BI space. We recommend that you assess them.

We have regularly used **New Relic** hosted performance monitoring with Ruby on Rails systems in development and production. The combination of fast setup and comprehensive reporting has proven extremely valuable in troubleshooting performance. We are now seeing good results from the New Relic monitoring services for Java and .NET systems.

Powershell is as important tool for managing Windows servers and applications. Built into Windows 2008 and Windows 7, Powershell allows Unix-like scripting and automation across a server farm. Scripts can be executed on remote machines, and a single command can manage hundreds of machines at once. Powershell scripts can deploy and configure applications and operating system components, and can be extended by writing .NET "commandlets."

Gradle is an attempt to bring sanity to the enterprise build space by marrying best-of-breed tools with cutting edge techniques. Gradle allows you to interact with your existing Maven repositories, but adds scriptability to your builds with a clean domain specific language.

Interesting Tools

Fog – Cloud services library for Ruby to administer compute, CDN, DNS and storage using a common API – **http://fog.io**

MCollective – MCollective supports automated parallel command execution on large clusters of servers. Instead of addressing servers by hostnames or IP addresses, it allows querying by server facts – reducing a source of maintenance problems. – **http://puppetlabs.com/mcollective/introduction/**

Midje – Tool for readable tests in Clojure – **https://github.com/marick/Midje/wiki**

Play Framework – Framework for creating web applications with Java and Scala – **http://playframework.org/**

Platforms

Mobile web was in our trial ring on previous radars, but we've moved it into adopt in recognition of the fact we have created many mobile web applications. We believe this is the right way to create web content for mobile devices.

HTML5 includes features that allow control and storage of offline data within the browser using client side JavaScript. These features allows creation of **offline mobile web applications** in a cross platform way that would have previously required installed applications. For instance an application that can download articles for reading later or a data capture application that can work offline and upload when you are online. While the standard is not finalized yet, support for these offline features is available and ready for use in the WebKit based browsers found on iOS, Android and newer Blackberry phones.

Tablet devices provide a new model of computing. The next generation of tablets show the potential for new interaction paradigms, and we expect interest and innovation to continue to escalate.

Ubiquitous computing is tricky term as it covers many different ideas. What we find interesting and exciting at the moment is that both consumer and specialist mobile devices are increasingly based on commodity operating systems such as Android or iOS. This means that in many cases, software can be developed by organizations themselves, opening the door to innovative new applications without requiring expensive niche skills. Lower price points for the hardware also make this area more accessible, especially with peripherals like payment card readers, PIN key pads and high quality bar code scanners becoming available for both Android and iOS devices. When combined with features already available on these consumer devices, whole new ways of working open up.

OAuth is a web-friendly, lightweight standard for authorization that allows a user to share private resources between internet services, e.g., allowing your favorite social networking site to access your photos from your favorite photo sharing site. OAuth is simple, avoids password proliferation, and allows a service to grant bare minimum privileges. If you are exposing your application's data in a lightweight, web-friendly manner you should strongly consider using OAuth as your standard for authorization.

Charles Nutter and the JRuby team continue to improve **JRuby** at a frantic pace. It is fast and they place massive importance on keeping their ecosystem up-to-date, including DB adapters, gem management, and modern Rails deployment. Rails 3 + JRuby is an awesome platform. There really is no reason to not be using Ruby, one of our favorite languages, in the enterprise.

Amazon continues to evolve the **AWS** cloud with services such as RDB, making it even easier to engineer and deploy cloud-based applications. Not every AWS feature is as mature as EC2 and S3, so you should carefully evaluate which AWS components to use. We feel comfortable recommending AWS where elasticity or on-demand computing are required.

Cloud Foundry is an open source Platform as a Service that can be deployed in your own data center or hosted by VMWare. At present Cloud Foundry supports Java/ Spring applications, Rails, Sinatra, Grails and Node.js. Additional services include MongoDB, MySQL and Redis. The platform seems to be enjoying active development with the recent addition of Scala and Lift support. Cloud Foundry is an interesting addition to the growing list of PaaS solutions. It is not clear what the relationship between vFabric and Cloud Foundry will be going forward.

Platforms *continued*

Heroku is a beautifully simple Platform as a Service (PaaS). Although Heroku began as a Ruby on Rails platform, it is evolving to support a variety of languages and web frameworks, most recently Clojure. Heroku uses a standard stack and deploys applications with a simple Git push. Heroku's recent acquisition by Salesforce.com has not diminished its service quality.

A continuing cause of delivery problems lies in the use of **Java Portal Server** packages. These problems occur in both open source and commercial portal platforms. The promised productivity of these platforms is hindered by their complex and unwieldy programming models and difficulty in automating deployment, data migration, and tests. Although product demos are compelling, the base features of portal products are often a poor fit for real web applications, while the extra advertised features such as single sign-on or search are usually already served by existing, targeted, enterprise assets.

GWT is a reasonable implementation of a poor architectural choice. GWT attempts to hide many of the details of the web as a platform by creating desktop metaphors in Java and generating JavaScript code to implement them. First, in many ways, JavaScript is more powerful and expressive than Java, so we suspect that the generation is going in the wrong direction. Secondly, it is impossible to hide a complex abstraction difference like that from event-driven desktop to stateless-web without leaky abstraction headaches eventually popping up. Third, it suffers from the same shortcomings of many elaborate frameworks, where building simple, aligned applications is quick and easy, building more sophisticated but not supported functionality is possible but difficult, and building the level of sophistication required by any non-trivial application becomes either impossible or so difficult it isn't reasonable.

Previously our advice has been to tread carefully when using the **WS-*** stack beyond the basic profile. Given the progress and acceptance of simpler web-as-platform techniques such as REST and OAuth and the known issues with the WS-*, it should only be used cautiously.

Languages

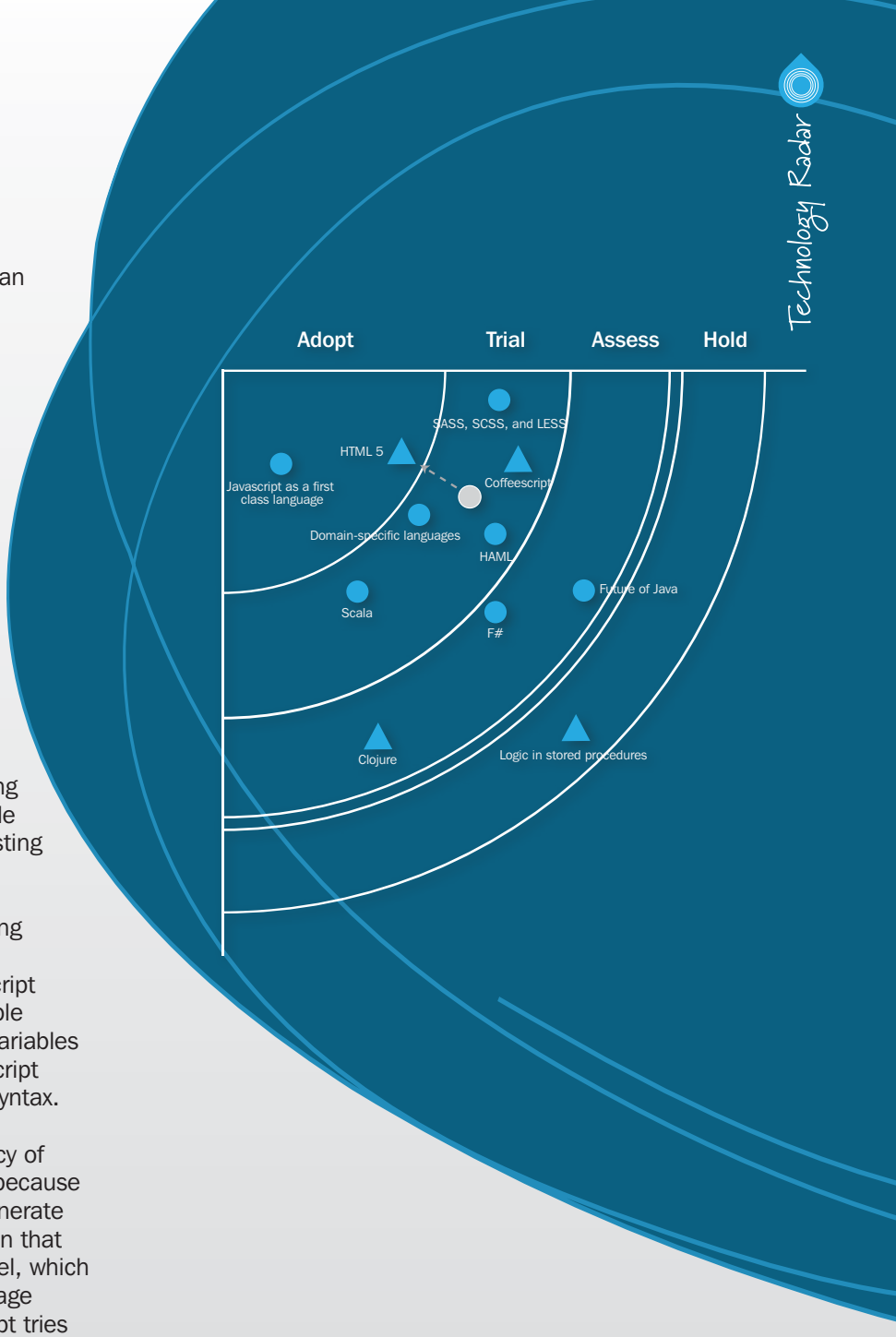
While **HTML5** is an evolving standard, many elements have reached the stage where they can be safely used in production to create both on and offline mobile web applications. Based on our projects we think HTML5 is ready to be adopted for mobile web applications. As the standard continues to evolve we expect HTML5 will become an increasingly viable alternative to native applications with the distinct advantage of being inherently cross platform.

Clojure is a dynamic, functional language that runs on the JVM. Although its roots are in Lisp, one of the oldest computer languages, it also embodies many modern programming concepts, including lazy evaluation and advanced concurrency abstractions. Clojure has spawned a vibrant community of programmers who are contributing a rich set of frameworks and tools. One example of these is Midje, an innovative spin on unit testing and mocking frameworks

JavaScript is a powerful, ubiquitous programming language with tricky and error prone syntax. **Coffeescript** fixes many of the warts of JavaScript in a clean, simple syntax that generates readable JavaScript. For example, creating true private variables in JavaScript is a syntactic nightmare; CoffeeScript generates the technically correct but hideous syntax.

Some readers may be confused by our advocacy of Coffeescript given our general dislike for GWT, because on the surface they seem similar: tools that generate JavaScript. However, it is the level of abstraction that differs. GWT has an elaborate component model, which tries to hide details about the underlying language (JavaScript) and platform (the web). Coffeescript tries to make it easier to write proper JavaScript, avoiding pathological but default “features” of JavaScript, and does not build a layer that tries to insulate you from the platform.

It is startling to us that we continue to find new systems in 2011 that implement significant **business logic in stored procedures**. Programming languages commonly used to implement stored procedures lack expressiveness, are difficult to test, and discourage clean modular design. You should only consider stored procedures executing within the database engine in exceptional circumstances, where there is a proven performance issue.



References

Backbone.js http://documentcloud.github.com/backbone/
Business logic in stored procedures http://bit.ly/storedprocedure
Cloud Foundry http://cloudfoundry.org/
CoffeeScript http://jashkenas.github.com/coffee-script/
Continuous Delivery http://bit.ly/delivery-vs-deployment
Cross platform mobile http://martinfowler.com/bliki/CrossPlatformMobile.html
Event Sourcing http://martinfowler.com/eaaDev/EventSourcing.html
Feature Branching http://martinfowler.com/bliki/FeatureBranch.html
Gradle http://gradle.org/
JQuery Mobile http://bit.ly/jQueryMobile
New Relic http://newrelic.com
Offline mobile Webapps http://diveintohtml5.org/storage.html
Selenium 2 testing of mobile websites http://slidesha.re/selenium2
Sonar http://sonarsource.org/
Test Load Balancer http://test-load-balancer.github.com/

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, ThoughtWorks Studios, offers tools to manage the entire Agile development lifecycle through its Adaptive ALM solution™, comprised of Mingle®, Go® and Twist®. ThoughtWorks employs 1,700 professionals to serve clients from offices in Australia, Brazil, Canada, China, Germany, India, 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, call us.