Technology Radar

http://www.thoughtworks.com/radar

Prepared by the ThoughtWorks
Technical Advisory Board

August 2010

ThoughtWorks®

What's new?

Since the last publication of the Tech Radar:

- Continuous Deployment, Next-gen test tools, NoSQL, and Facebook as a business platform moved from assess to trial
- Visualization & metrics, Emergent Design, Android, and JVM as a platform moved from trial to adopt
- Azure moved from hold to assess
- 18 new items entered the Radar. These are represented as triangles.

Introduction

The ThoughtWorks Technical Advisory Board consists of a group of senior technical 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. With this in mind the content provided in this document is kept at a summary level, leaving it up to the reader to pursue more detailed knowledge as the need arises.

The goal of the radar is conciseness, so that its target audience understands it quickly. To that end, it is graphical in nature. Radar items are grouped into techniques, tools, languages and platforms. Some radar item 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:

- Hold: Of interest to ThoughtWorks and others in the industry, but, in our opinion, not ready to invest significant time and resources to build experience.
- Assess: Worth exploring with the goal of understanding how it will affect your enterprise.
- 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.
- Adopt: Industry has finished trial and found proper patterns of usage, or we feel strongly that the industry should be adopting it now, rather than going through a gradual adoption approach.

As we look at each quadrant in detail, we try to show the movement that each item has taken since the last publication of the radar. New items are represented as triangles, while items that were on the last radar are represented as circles.

Contributors

The ThoughtWorks Technical Advisory Board is comprised of

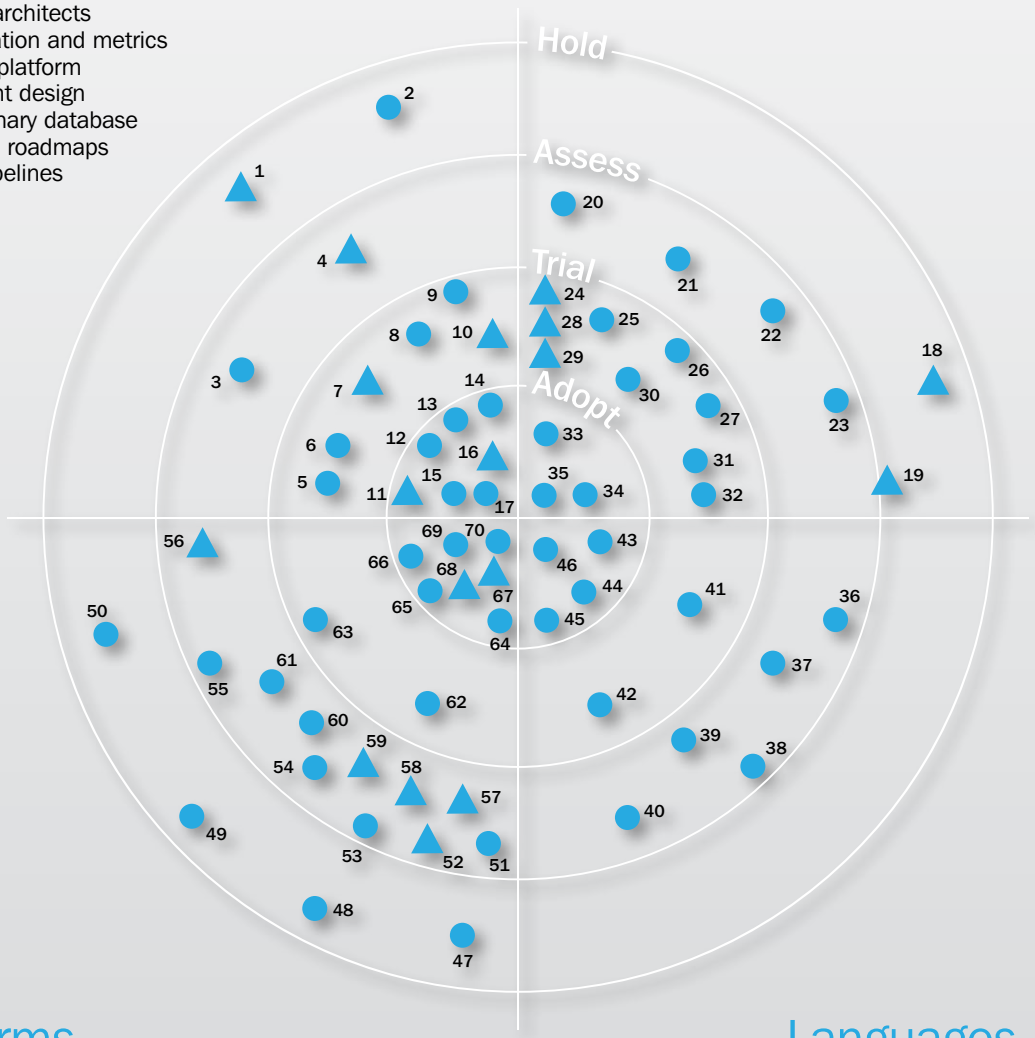
Rebecca Parsons (CTO)
Martin Fowler (Chief Scientist)
Nick Hines (CTO Innovation)
Graham Brooks
Ian Cartwright
Erik Doernenburg
Jim Fischer
Neal Ford
Ajey Gore
Wendy Istvanick
Mike Mason
Cyndi Mitchell
David Rice
Ian Robinson
Pramod Sadalage
Samir Seth
Jim Webber
Hao Xu
Jeff Norris.

Techniques

1. Database based integration
2. Scrum certification
3. Incremental data warehousing
4. DevOps
5. Polyglot programming
6. Automation of technical tests
7. Capability modeling
8. Service choreography
9. Continuous deployment
10. Evolutionary architecture
11. Coding architects
12. Visualization and metrics
13. Web as platform
14. Emergent design
15. Evolutionary database
16. Platform roadmaps
17. Build pipelines

Tools

18. ESB
19. Intentional
20. Cross mobile platforms
21. Github
22. Restfulie
23. RDF triple stores
24. Apache camel
25. Next gen test tools
26. NoSQL
27. Neo4j
28. Message buses without smarts
29. Puppet
30. mongoDB
31. Mercurial
32. Git
33. Squid
34. ASP.NET MVC
35. Subversion



Platforms

47. Rich internet applications
48. GWT
49. IE8
50. WS-* beyond basic profile
51. Azure
52. Mobile Web
53. Google App Engine
54. Application appliances
55. Google as corporate platform
56. GPGPU
57. App containers
58. OAuth
59. RDFa
60. Location based services
61. iPad
62. EC2 & S3
63. Facebook as business platform
64. JVM as platform
65. iPhone
66. Android
67. KVM
68. Atom
69. ALT.NET
70. IE6 End of Life

Languages

36. Java language end of life
37. F#
38. Scala
39. Clojure
40. HTML 5
41. DSL's
42. Groovy
43. C# 4.0
44. JRuby
45. JavaScript as a first class language
46. Ruby

Techniques

It is rare for enterprise software to be developed in isolation, with no dependencies on other systems, deployed into production environments without support from operations teams, and without considering the strategic goals of the business. Yet, in our day to day experience delivering software with client teams, we continue to see development and delivery practices that overlook these fundamental concerns.

For those organizations that are required to integrate systems, many continue to use a common database, sharing data between applications through the database tier. In many cases this has become an established and accepted architectural pattern: **database based integration**. The side affect of such an approach is greater coupling of database schemas, release schedules, performance and quality of service across applications.

DevOps is a new movement seeking to achieve the business need for rapid delivery of software products while maintaining the stability of live environments. It uses two approaches: first, promoting closer collaboration between development and operations; second, applying practices shared with agile (collaboration, automation, simplicity, etc) to operations processes such as provisioning, change management, and production monitoring. It encompasses culture, processes, and tools - all supporting better communication, faster feedback and delivery, and more predictable outcomes.

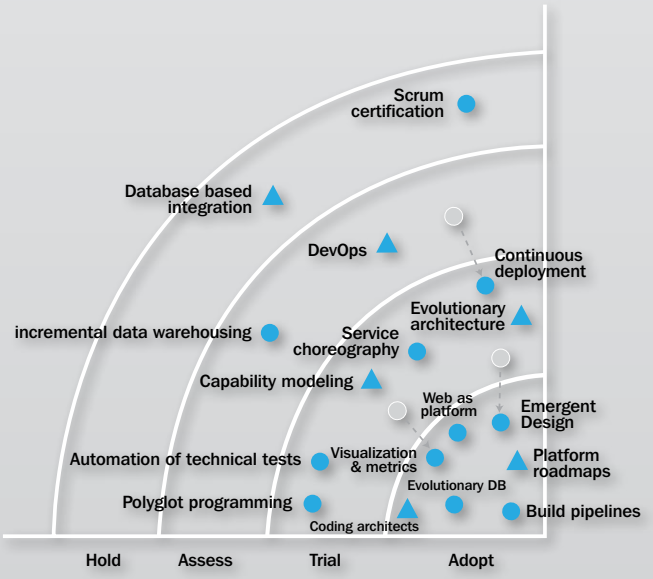
One principle of agile software development is the notion of the last responsible moment. This notion applied to architectural considerations is controversial among traditional architects. We believe that, given properly articulated principles and appropriate test suites, architectures can evolve to meet the changing needs of a system, allowing for architectural decisions to be made at the last responsible moment without compromising the integrity of the system. We call this approach **evolutionary architecture**, in that we allow the architecture to evolve over time, always respecting the architectural guiding principles.

Initiatives that span multiple projects require shared understanding of the business context, operating model, and strategic goals of an organization, as well as any existing technical, organizational and process constraints impinging on planning and design activities. As part of our evolutionary approach to enterprise architecture, we use business **capability modelling** to create lightweight hierarchical models of the business functions that are an essential part of an organization's needs and goals. Capabilities describe an organization's operating model in terms of goals and competencies (what is to be done), rather than implementation specifics (how things are done). Whereas business architecture models based on people, process or technology are contingent, volatile and often short lived, and therefore ill-suited to the long-term planning needs of the organization, capability models provide a description of the business context that is stable enough to serve as a basis for identifying and prioritising technology and process initiatives.

Integrated business processes now routinely span multiple systems and even enterprises. This raises the question of how these processes should be coordinated. In our experience centralized orchestration solutions often fail to deliver the promised benefits. They are costly to implement, and because they maintain application state on behalf of many consumers, they are often difficult to scale. This has lead us to prefer **service choreography**, where independently distributed participants collaborate according to an application protocol. Using the Web as platform, hypermedia-driven application protocols allow us to implement integrated business processes that are easy to evolve and easy to scale.

We strongly believe that all software delivery organizations need to be making use of **automated technical tests**. This sort of test spans failover testing, performance testing and soak testing among others; these activities can start early in a project's life-cycle and continue through to maintenance. The common practice of waiting until near the end of a project is fraught with risk with little time available to find and fix problems. For example the requirement for a comprehensive production-like environment before the start of performance testing is a dangerous fallacy, we can discover bottlenecks, track performance trends and test our performance tests, without waiting for a perfect environment.

Google maps has led the way in bringing mapping mainstream. But businesses, governments and non-profit organizations are now learning to use these **location based services** to communicate more effectively with customers. With other mapping services providers getting into the act, there is going to be a proliferation of applications built around mapping targeted at customers who are now much more map-savvy.



Tools

In today's connected systems environments almost all new development needs to integrate with existing applications and services. In conjunction with our adoption of simple message buses and integration techniques at the edges of a system, we have successfully used small libraries such as **Apache Camel** to perform the protocol bridging, message transformation and message routing tasks common to such integrations. Camel's fluent Java interface, unit testing support and connectors for many different transports and message formats provide for an effective anti-corruption layer when implementing distributed applications.

While there has been much publicity around Apple's squashing **cross-platform development** options for the iPhone and iPad, there are still perfectly valid options. PhoneGap and Appcelerator Titanium's approach is to provide a native compatibility layer for all the major mobile platforms to your Web standard HTML+CSS+JS application.

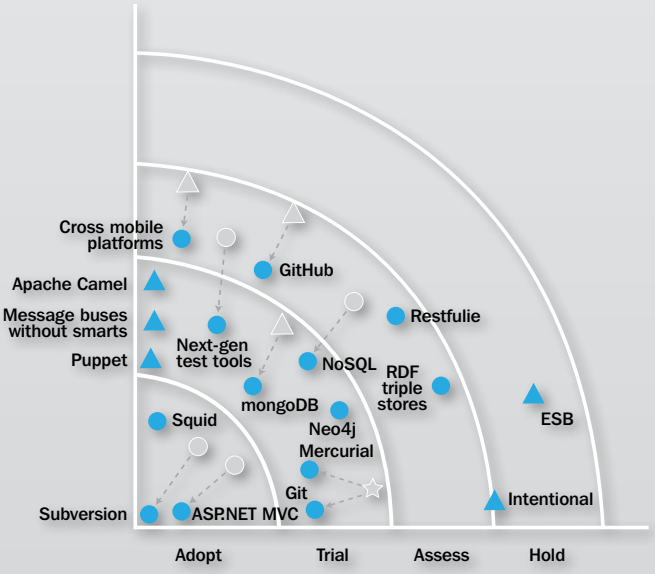
In previous radars we recommended Distributed Version Control (DVCS) tools in general while mentioning **Git** and **Mercurial** in particular. In this edition we narrow our recommendation to only Mercurial and Git as these two have become the clear front-runners. Due to its success and the associated network effect GitHub remains the recommended option for enterprises that want to interact with the open source community.

In the last radar we placed the **Google Web Toolkit (GWT)** on hold and tried to provide a few reasons for that decision. As it turned out the conciseness of the text didn't allow us to adequately make our points so that they were not misunderstood. We are interested in a discussion but our opinion about the suitability and usability of GWT has still not changed.

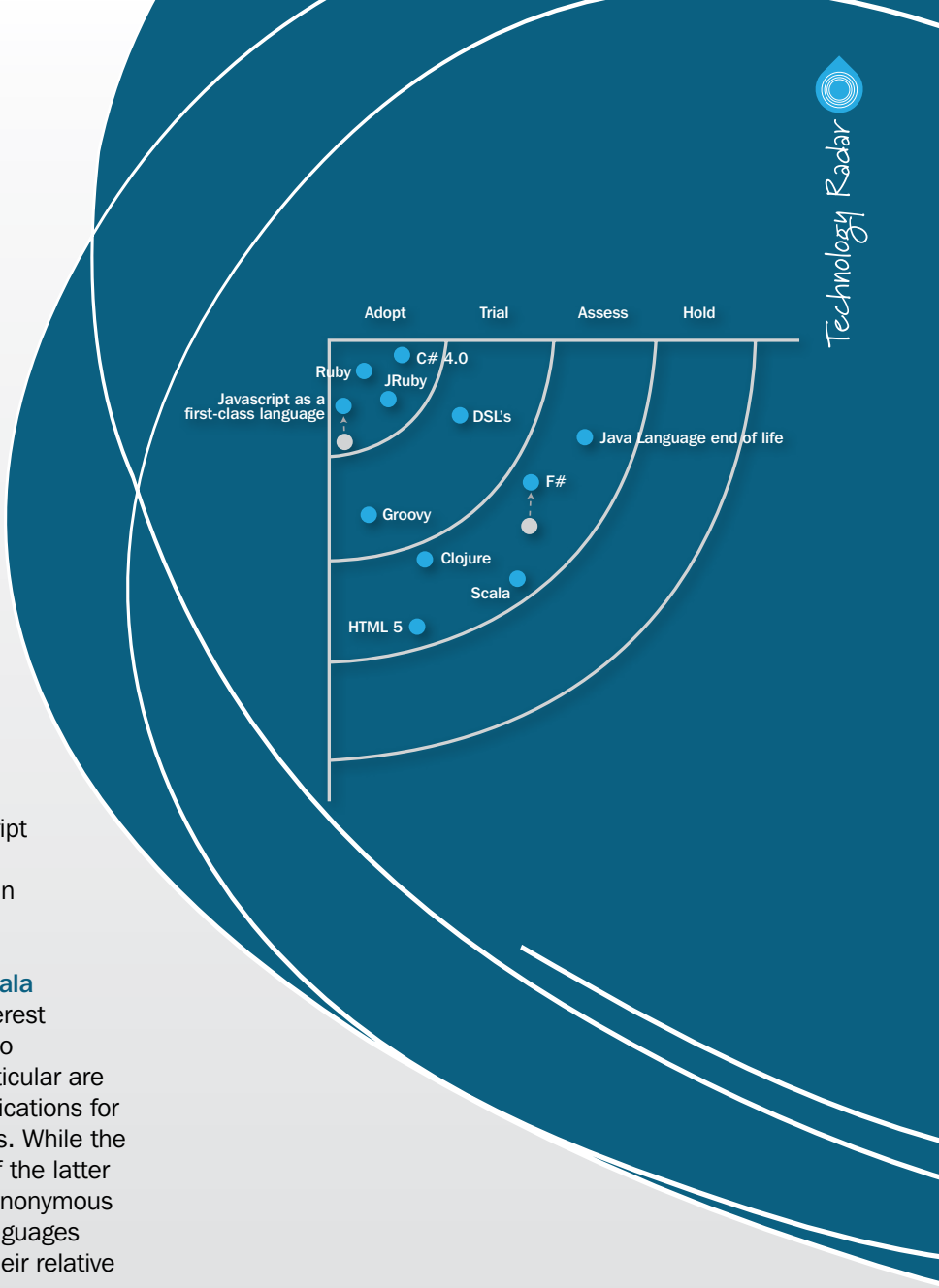
Puppet is a free, open source data center automation tool for managing changes to your production and production-like environments. Using Puppet, you can keep the configuration of your environments in version control and push changes out to your systems in a controlled, automated fashion. Infrastructure automation tools like puppet have the benefits of reducing manual effort allowing ops to focus on higher priorities, providing consistency and repeatability by reducing waste eliminating environmental differences between test and production environments.

NoSQL is about scale, massive datasets, cloud data, social network data, data in buckets, data in graphs i.e. a range of use cases for which "traditional" SQL databases may not be the optimal choice. Unravelling NoSQL and trying to explain what it is and why you should or should not be interested in it is difficult as the term covers a wide range of technologies, data architectures and priorities and represents as much a movement or a school of thought as it does any particular technology. Types of NoSQL technologies include key-value, column and object stores as well as document, graph and XML databases.

ThoughtWorks has been working with **Intentional Software** for the past several years, and we are thrilled at the recent limited availability and production use of the Intentional Domain Workbench. We believe this technology represents a radical departure from the traditional software development approach. We place this technology in the assess ring, since we believe that it is time to begin exploring the application of Intentional's technology in proofs of concept.



Languages



The maintainability, testability and readability of **JavaScript** is a very significant contributor to the productivity of teams producing Web-based applications and sites. ThoughtWorks believes JavaScript deserves to be treated as a first class language, viewing it as second class citizen has become an excuse for a whole series of bad practice we would not tolerate in Java or C#. We need to use the same kind of tools (e.g. unit testing) and approaches (e.g. refactoring) as we'd use for any other production language. V8 and other JavaScript engines are raising the bar on performance, while Flash & Silverlight seem to be losing momentum to HTML5 + JavaScript in areas where a rich client-like experience is required. This is good news for all interested in open standards on the Web.

The functional languages **F#**, **Clojure** and **Scala** still reside in the assess ring of the radar. Interest in functional languages continues to grow. Two characteristics of functional languages in particular are driving this interest, immutability with its implications for parallelism and functions as first class objects. While the introduction of closures to C# brings some of the latter capability, functional languages are almost synonymous with immutability. The placement of these languages within the assess ring indicates our view of their relative maturity and appropriateness.

F#, based on OCaml, is fully supported within the Visual Studio toolset. F# includes support for objects and imperative constructs in addition to functional language constructs in a natural way. Scala, like F#, combines the object and functional paradigms, although the syntax of Scala is more Java-like. Clojure began as a JVM language and is now available on the .NET CLR. Clojure does allow for mutable state although it has an extensive set of immutable persistent data structures, all supporting multi-threaded applications. There are many similarities between these three languages, but at the moment we believe F# and Clojure to be better suited to most organizations for assessing than Scala. More work clearly needs to be done to validate this assertion.

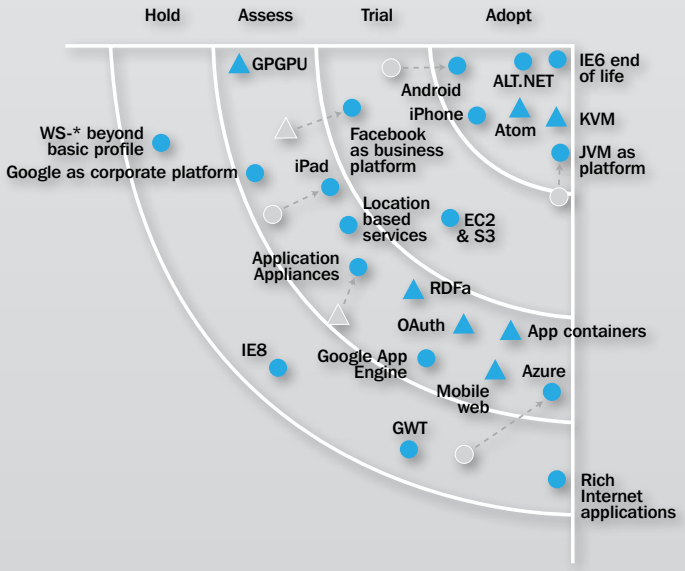
Platforms

HTML 5 continues to be the preferred choice for developing complex Web-based applications, with features including improved integration of rich audio and video content, client-side storage, better document structure, Web sockets and offline use. Safari, Chrome, Firefox and Opera each support significant subsets of the proposed standards, with support coming in Internet Explorer 9. HTML 5 is likely to remain in draft for some time to come, however; early adopters may wish to reflect on the bleakly comedic saga of two separate groups attempting to drive its evolution.

RDFa, a mechanism for attaching meaningful vocabularies to HTML content that is being quickly and widely adopted by content providers, is the first mainstream success to arise from the Semantic Web stack. RDFa enables tools ranging from custom point integrations to Google spiders to more richly understand your Web content. If you would like to quickly open up your content to a multitude of integration possibilities in a simple, cheap, standards-based fashion, we recommend you try RDFa.

While we are bullish on RDFa, we remain highly guarded on native **RDF triple stores** as a persistence mechanism. The leading available triple stores vary greatly in their capabilities, capacity, and performance characteristics. If you are exploring the use of a triple store, you must do extensive testing to make sure the triple store fits your needs.

One of the foundational technologies of the Web as platform, **Atom** is an extensible data syndication format with broad tool support in almost all languages. In conjunction with the Atom Publication Protocol, Atom comprises a lightweight platform for publishing and consuming data with high quality-of-service guarantees. Atom-based solutions trade scalability for latency, however, making Atom often inappropriate for very low-latency scenarios.



OAuth is a Web-based authorization protocol that allows applications to access a user's secured resources in another application without the user having to share their private security credentials. Now an RFC, OAuth represents a significant standards-based attempt to improve privacy and security for Web browser and machine-based access to distributed Web resources. Library support is patchy and adopters can expect to spend some time wrangling their code to achieve true interoperability. OAuth 2.0 is due towards the end of 2010, with specific flows for Web applications, desktop applications, mobile phones, and household devices. Because OAuth 2.0 is not backwardly compatible with version 1 and the implementation challenges around the current version, OAuth is still in the assess ring.

AWS is the most mature and broadest of the current cloud offerings providing scalable services for computation **(EC2)**, storage (**S3** & SBS), databases (SimpleDB & RDS), messaging (SQS & SNS), etc. The list of services provided by AWS continues to expand rapidly with new services being added on an almost monthly basis, (http://bit.ly/90887v). While existing applications can be deployed on AWS through the use of Amazon Machine Images the full benefits of this platform will come from applications that are developed to take advantage of AWS. The usage based billing model adopted by AWS allows organizations to scale applications without large upfront investment and avoid the overhead cost of under utilized hardware.

The **iPhone** changed the face of the mobile phone. The **iPad** has the potential to radically alter the way users interact with and consume Web-based resources and applications and will spawn a plethora of similar tablet devices. The addition of wireless application distribution in IOS4 allows organizations to securely host and distribute in-house applications without using the App Store, overcoming one of the main barriers to corporate adoption. IOS4's introduction of multitasking with applications running in the background has opened up new possibilities for enterprise applications, at the cost of extra battery usage.

The use of **GPUs** for computing offers efficiencies and performance for certain classes of problems that would be prohibitively expensive for more traditional hardware. Problems that fit Single Instruction Multiple Data (SIMD) processing models can gain significant advantages at the cost of difficult learning curves using specialized APIs. OpenCL, CUDA from NVidia and DirectCompute from Microsoft offer developers access to General-purpose computing on graphics processing units (GPGPU).

References

http://www.infoq.com/presentations/agilists-and-architects

http://martinfowler.com/ieeeSoftware/enterpriseArchitects.pdf

http://intentsoft.com/

http://www.infoq.com/presentations/Intentional-Software-at-Work

http://clojure.org/

http://fsharp.net

http://www.scala-lang.org/

ThoughtWorks is a global IT consultancy

We deliver custom applications and provide consulting grounded in reality; we help organizations become efficient through Agile and Lean practices and principles. By hiring exceptional people, we can solve our clients' biggest and most pressing problems. All of our services are offered both on and offshore, and are delivered with pride and passion.