

Dion Hinchcliffe's Blog - Musings and Ruminations on Building Great Systems

Agile Methods, Enterprise Architecture, 2.0 Services, and Web Development

10 Must-Know Topics For Software Architects In 2009

In the last year or so, after quite a lull, the software architecture business has gotten rather exciting again. We're finally seeing major new topics emerging into the early mainstream that are potential game-changers, while at the same time a few innovations that have been hovering in the margins of the industry are starting to break out in a big way.

The big changes: The hegemony of traditional 3 and 4-tier application models, heavyweight run-time platforms, and classical service-oriented architecture that has dominated for about a decade is now literally being torn asunder by a raft of new approaches for designing and architecting applications.

These might sound like incautious words but major changes are in the air and architects are reaching out for new solutions as they encounter novel new challenges in the field. As a consequence, these new advances either address increasingly well-understood shortcomings of existing approaches or add new capabilities that we haven't generally focused on before but are becoming increasingly important. A few examples of the latter include creating reusable platforms out of applications from the outset (the [open API story](#)) or cost-effectively creating architectures that can instantly support global distribution, hundreds of terabytes of data, and tens of millions of users. There are others that we'll explore throughout this post.

These innovations are hallmarks particularly of the largest systems being built today (which are running into unique challenges due to scale, performance, or feature set) though these software advances are also moving across the spectrum of software from everyday corporate systems and Internet applications to new mobile devices and beyond, such as the emerging space of social networking applications.

Mainstays of application architecture such as the relational database model, monolithic run-times, and even deterministic behavior are being challenged by *non-relational systems*, *cloud computing*, and new *pull-based systems* where consistency and even data integrity sometimes take a backseat to uptime and performance.

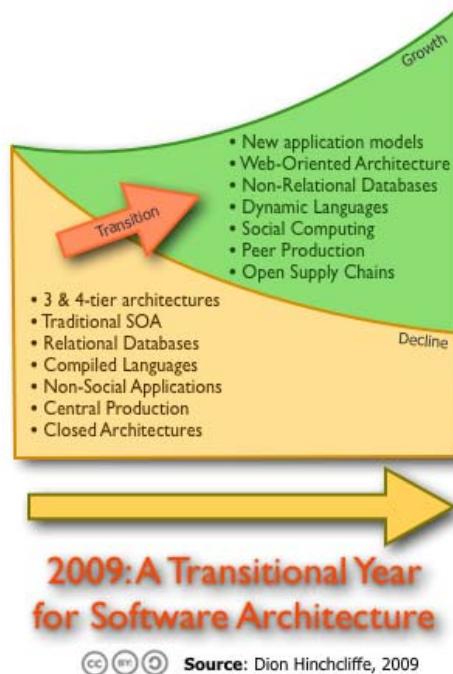
Let's also not forget about Web 2.0 approaches and design patterns which are becoming ever more established in online applications and enterprise architecture both. *Social architectures*, *crowdsourcing*, and *open supply chains* are becoming the norm in the latest software systems faster than expected in many cases. Unfortunately, as a result, the architectural expertise needed to effectively leverage these ideas is often far from abundant.

To try to get a handle on what's happening and to explore these emerging topics, I've been doing conference talks lately about the transformation of software architecture that we're beginning to see in so many quarters these days and generally finding consensus that the exciting days of architecture are back, if they ever left. Now it's up to us to begin the lengthy process of taking many of these ideas into our organizations and integrating them into our thought processes and architectural frameworks and bringing them to bear to solve problems and provide value. As one software architect came up and asked me recently, "*How do I get my organization to understand what's happening out there?*" This is an attempt at addressing that question.

Here's a list of the most important new areas that software architects should be conversant in and looking at in 2009:

10 Must-Know Topics for Software Architects in 2009

1. **Cloud Computing.** This one is easy to cite given the amount of attention we're seeing in the blogosphere and at conferences, never mind the (considerable) number of actual users of popular cloud services such as [Amazon EC2](#). While the term doesn't have an exact definition, it covers the gamut of utility hosting to [Platform-as-a-service \(PaaS\)](#). I've covered [cloud computing on ZDNet in detail before](#) and [broken down the vendor space](#) recently as well. While the economics of cloud computing can be extremely compelling and there is undoubtedly a model that will fit your particular needs, cloud computing is also ground zero for the next generation of the famous OS platform wars. Walk carefully and prototype often to get early competency in an architectural advance that will almost certainly change a great deal about the software business in the near future.
2. **Non-relational databases.** Tony Bain over at [Read/Write Web](#) recently asked "Is The Relational Database Doomed?" While it's far too soon to declare the demise of the workhorse relational database that's the bedrock of so many application stacks, there a large number of promising alternatives emerging. Why get rid of the traditional relational database? Certain application designs can greatly benefit from the advantages of document or resource-centric storage approaches. Performance in particular can be much higher with non-relational databases; there are often surprisingly low ceilings to the scale of relational databases, even with clustering and grid computing. And then there is [abstraction impedance](#), which not only can create a lot more overhead when programming but also hurts run-time performance by maintaining several different representations of the data at one time during a service request. Promising non-relational solutions include [CouchDB](#), which I'm starting to see in more and more products, as well as [Amazon SimpleDB](#), [Drizzle](#) (from the MySQL folks), [Mongo](#), and [Scalaris](#). While many applications will continue to get along just fine with relational databases and object-relational mapping, this is the first time that mainstream database alternatives are readily available for those that are increasingly in need of them.
3. **Next-generation distributed computing.** An [excellent story today in the New York Times](#) about [Hadoop](#) provides a good backdrop on this subject: New distributed computing models are moving from the lab and becoming indispensable for providing otherwise difficult to harness computing power when connected to previously unthinkable quantities of data. While traditional request-response models that are the mainstay of network-oriented computing are important, so increasingly are effective ways to process the huge amount of data that are now common in modern software systems. Watch this [video interview with Mark Risher and Jay Pujara](#) at Yahoo that discusses how Hadoop "*enables them to slice through billions of messages to isolate patterns and identify spammers. They can now create new queries and get results within minutes, for problems that took hours or were considered impossible with their previous approach.*" While Hadoop has considerable momentum, other similar offerings include the commercial [GridGain](#) and open source [Disco](#) and there are many others.



E-mail | Twitter

TamTamy

Social
Networking
designed
for
your
business.

www
.tamtamy
.com

New: Subscribe via e-mail

Enter your email address:

Subscribe

Delivered by [FeedBurner](#)

my most complete write-up is [here](#). In short, the premise is that RESTful architectures (and the architecture stack above and around it including data representation, security, integration, composition, and distribution) are a more natural, productive, and effective way to build increasingly open and federated network-based applications. The WOA debate has raged for a while now since it became a hot topic last year but the largest network on the world has cast its vote and WOA is the way that the Web is going by and large; WOA-based applications just align better to the way the network itself inherently works. In my opinion, it is a much better way to create service-oriented architecture for almost all requirements, resulting in more supple and resilient software that is less difficult and expensive to build and maintain. For enterprises

- considering the move to WOA, here is [good overview I did a short while back](#) about the issues and the evolution of SOA.
- Mashups.** David Linthicum [wondered today in Infoworld](#) where the mashups have gone, clarifying that he believed they had become integral to SOA and for delivering value in enterprise architecture. In reality, while mashups are extremely common in the consumer space, to the point that it's just an every day application development activity, the tools and concepts are just now ready for prime-time in business. I've previously called mashups one of the [next major new application development models](#) and that's just what's happened. Mashups were also prominent in my [Enterprise Web 2.0 Predictions for 2009](#) (item #7). If you're not studying mashup techniques, Michael Ogrinz's [Mashup Patterns](#) is an excellent place to start studying how they impact software architecture.
 - Open Supply Chains via APIs.** I find the term *open APIs*, which an increasing [body of evidence](#) shows are an extremely powerful model for cross-organization SOAs, to be confusing to the layperson so I've begun calling them "*open supply chains*." Opening up your business in a scalable, cost-effective manner as a platform for partners to build up on is one of the most powerful business models of the 21st century. However, there seems to be a large divide between native-Web DNA companies and traditional organizations in understanding how important this is (it's increasingly mandatory in order to compete online). All evidence so far points to this as one of the most important, though potentially difficult, things to get right in your architecture. Security, governance, scalability, and ease-of-consumption are all major subject areas and our enterprise architectures and SOAs must be ready for this business strategy as more and more organizations open up. Here's my recent ["state of the union"](#) on open APIs.
 - Dynamic Languages.** Though dynamic languages have been popular on the Web since Javascript and Perl first arrived on the scene, it's only been recently that it's become acceptable to develop "real" software with them. .NET and Java are still extremely compelling (and common) platforms for writing and running application code but it's dynamic languages like Ruby, Python, PHP, and now Erlang that are getting all the attention these days. Why is this? As I explored in a [detailed comparison](#) a while back, a trade-off in run-time performance has generally been found to enable a large boost in productivity by virtue of what this lets dynamic languages accomplish. It also doesn't hurt that a lot of work has gone into newer dynamic languages to make them extremely Web-friendly, which is now one of the most common use cases for any programming language. Dynamic languages have architectural trade-offs of course, like any technology, though increasingly frameworks like Rails, CakePHP, and Grails are built on top of them which bring the latest best practices and design patterns, something that is not happening as frequently with older platforms. The tipping point has arrived however, and dynamic languages are beginning to take the center stage in a significant percentage of new projects. Software architects should be prepared.
 - Social computing.** Developers and software architects are often uncomfortable with [social computing](#) aspect of software systems today but [Reed's Law](#) has unequivocally demonstrated that the value of social systems is generally much higher than non-social systems. Or you could just look at the many popular applications out there that are driven by their social behavior and derive their (often enormous) value from the participation it entails. Whether this is YouTube, Facebook, Twitter, or thousands of other social applications (business and consumer both), the lesson is clear: Social architecture is an important new layer in the application stack and it I've since made it two entire quadrants of [my view of Web 2.0 in the enterprise](#) as a consequence. A List Apart has a great introduction to [The Elements of Social Architecture](#) and I've identified some of the core patterns for this in my Enterprise 2.0 mnemonic, [FLATNESSES](#). Finding a high-value place for social computing in our enterprise architectures will be essential for modern software efforts.
 - Crowdsourcing and peer production architectures.** Increasingly, the public network (the Web) has been used to enable potent [open business models](#) that are beginning to change the way we run our businesses and institutions. This started with open source software and has since moved to media and is now encroaching on a wide variety of industries. The models for doing this online require software architectures that can support this including architectural models for harnessing collective intelligence, moderating it, aggregating it, and protecting it and the users that provide it. As I wrote a couple of months ago in [50 Essential Strategies for Creating a Successful Web 2.0 Product](#), these architectures of participation create most of the value in the software systems that employ them. If you're not sure this is a software architecture issue, just look at Amazon's [Mechanical Turk](#) or [CrowdSound](#), that latter which is a widget that allows even end-users to dynamically include crowdsourcing into their applications. You can also read John Tropea's [new exploration of this topic](#) for an application layer viewpoint.
 - New Application Models.** The Semantic Web seems to be on the rise again and I've already covered Platform-as-a-service and mashups here, but in addition to these we are seeing entirely new application models cropping up in scale online. Whether these are Facebook applications, next-generation mobile apps (iPhone, Android, RIM, etc), OpenSocial or just the increasing prevalence of widgets and gadgets, the trend in the atomization of software (which was done still perhaps the best and most effectively so far in Unix) is reminding us that we still have new discoveries ahead of us. While these often seem trivial, aka applications as a feature, it's also increasingly clear that these are going to be here to stay and can provide considerable point value when they're designed correctly. Certainly for next-generation intranets and portals as well as the online "desktop", micro-applications which have to contend both with scale and with being useful and secure while embedded in other applications is increasingly on the radar. Know how they work, why they are so popular (there are tens upon tens of thousands of Facebook and OpenSocial applications alone) and learn how they can be used to provide real utility and every day value.

Any list of what is new and important in software architecture must be personal perspective so I invite you to add your own below in comments.

The WOA Application Stack



Source: Dion Hinchcliffe, 2009. <http://hinchcliffeandcompany.com>

posted on Tuesday, March 17, 2009 4:15 PM



Dion's Twitter Updates

follow dhinchcliffe at <http://twitter.com>

Dion's Speaking Calendar:



Sponsored Advertising



This work by <http://hinchcliffe.org> is licensed under a [Creative Commons Attribution-Share Alike 3.0 Unported License](http://creativecommons.org/licenses/by-sa/3.0/). Dion Hinchcliffe

Hinchcliffe and Company
The Hinchcliffe Advisory

Contact

[My E-mail](#)

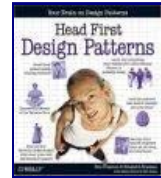
Tech News Read By Me

 [eWeek Enterprise Apps Center](#)

 [MSDN](#)

 [TheServerSide](#)

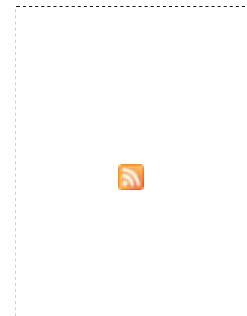
[Ward's Wiki Changes](#)



Head First Design Patterns:

Don't reinvent the wheel, use the most insanely great book on design patterns yet written.

My BlogMap!



Blogs nearby : [100+](#)

Powered by feedmap.net

