ThoughtWorks®

TECHNOLOGY RADAR *NOV '15*

Our thoughts on the technology and trends that are shaping the future

thoughtworks.com/radar

WHAT'S NEW?

Here are the themes highlighted in this edition:

DOCKER INCITES CONTAINER ECOSYSTEM EXPLOSION

Containerization, exemplified by Docker, is wildly popular in a growing number of organizations. The interest varies widely across and within organizations; our recommendations range from Assess to Adopt. The ecosystem (tools, platforms and techniques) is growing and maturing, further accelerating interest. Astute readers will note related topics across our radar, ranging from Docker as a development tool for managing dependencies to large cloud platforms such as Mesos and AWS ECS that use containers as their "unit of scaling".

MICROSERVICES AND RELATED TOOLS GAIN IN POPULARITY

Interest continues unabated around this architectural style, which transitively boosts interest in supporting tools and techniques: DevOps practices like containerization, lessons learned such as the perils of programming in your CI/CD tool, the maturity of service discovery tools, and so on. We expect to see even more growth and maturity in this space in the near future.

JAVASCRIPT TOOLING SETTLES TO MERELY CHAOTIC

We have highlighted the churn in the JavaScript tool space before, but the community is gradually calming and coalescing around some common practices. Teams are discovering the best combination (including none) for build tools and package management, and we hear less disagreement across teams on effective practices.

SECURITY IS EVERYBODY'S PROBLEM

Security is an issue that uniquely affects all roles across the software development lifecycle. We highlighted improvement in the security space in the last Technology Radar, and we're pleased to see teams baking security practices into their SDLC. In this edition we also highlight innovative approaches such as bug bounties, threat modelling, HSTS, TOTP and Let's Encrypt. We hope traction continues to improve in this space.

CONTRIBUTORS

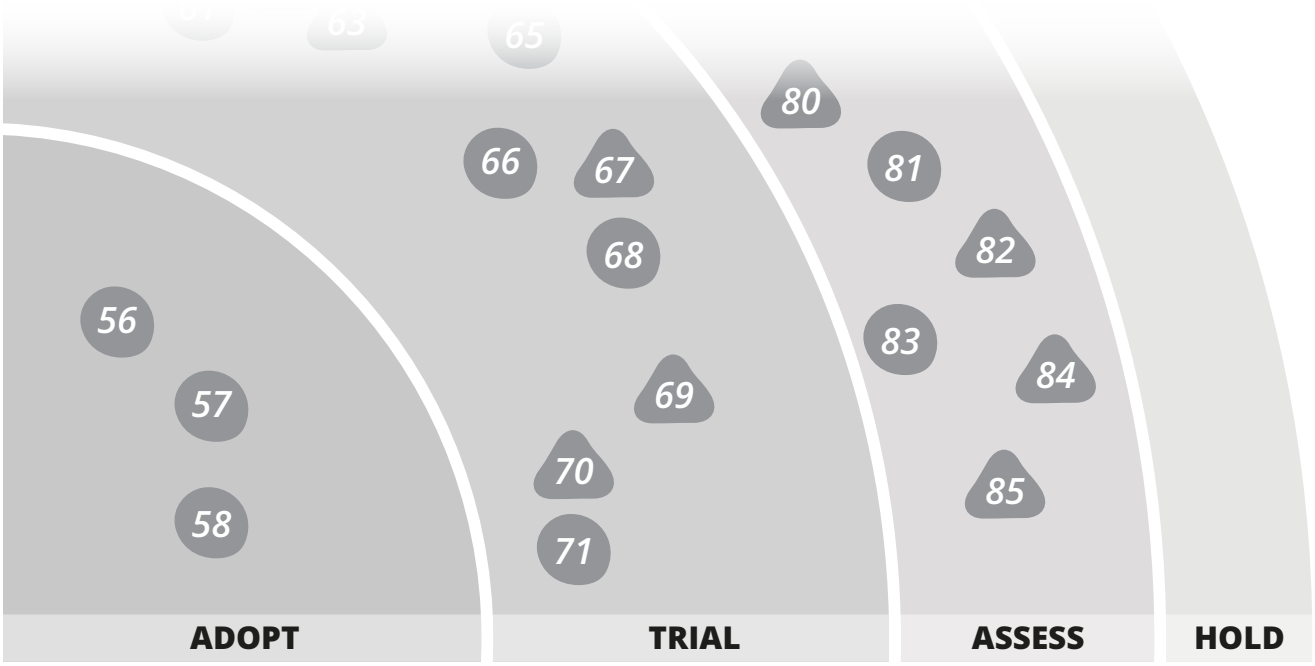
The Technology Radar is prepared by the ThoughtWorks Technology Advisory Board, comprised of:

Rebecca Parsons (CTO)	Claudia Melo	Ian Cartwright	Rachel Laycock
Martin Fowler(Chief Scientist)	Dave Elliman	James Lewis	Sam Newman
Anne J Simmons	Erik Doernenburg	Jonny LeRoy	Scott Shaw
Badri Janakiraman	Evan Bottcher	Mike Mason	Srihari Srinivasan
Brain Leke	Hao Xu	Neal Ford	Thiyagu Palanisamy

ABOUT THE TECHNOLOGY RADAR

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:



We feel strongly that the industry should be adopting these items. We use them when appropriate on our projects.

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.

Worth exploring with the goal of understanding how it will affect your enterprise.

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. 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 thoughtworks.com/radar/faq

THE RADAR

TECHNIQUES

ADOPT
1. Consumer-driven contract testing
2. Decoupling deployment from release new
3. Generated infrastructure diagrams
4. NoPSD
5. Products over projects
6. Threat Modelling

TRIAL
7. BEM new
8. BFF - Backend for frontends new
9. Docker for builds new
10. Event Storming new
11. Flux
12. Idempotency filter new
13. iFrames for sandboxing new
14. NPM for all the things new
15. Offline first web applications
16. Phoenix Environments
17. QA in production new

ASSESS
18. Accumulate-only data
19. Bug bounties new
20. Data Lake
21. Hosted IDE's new
22. Monitoring of invariants new
23. Reactive Architectures

HOLD
24. Gitflow new
25. High performance envy/web scale envy new
26. Microservice envy
27. Pace-layered Application Strategy
28. Programming in your CI/CD tool
29. SAFe™
30. Separate DevOps team

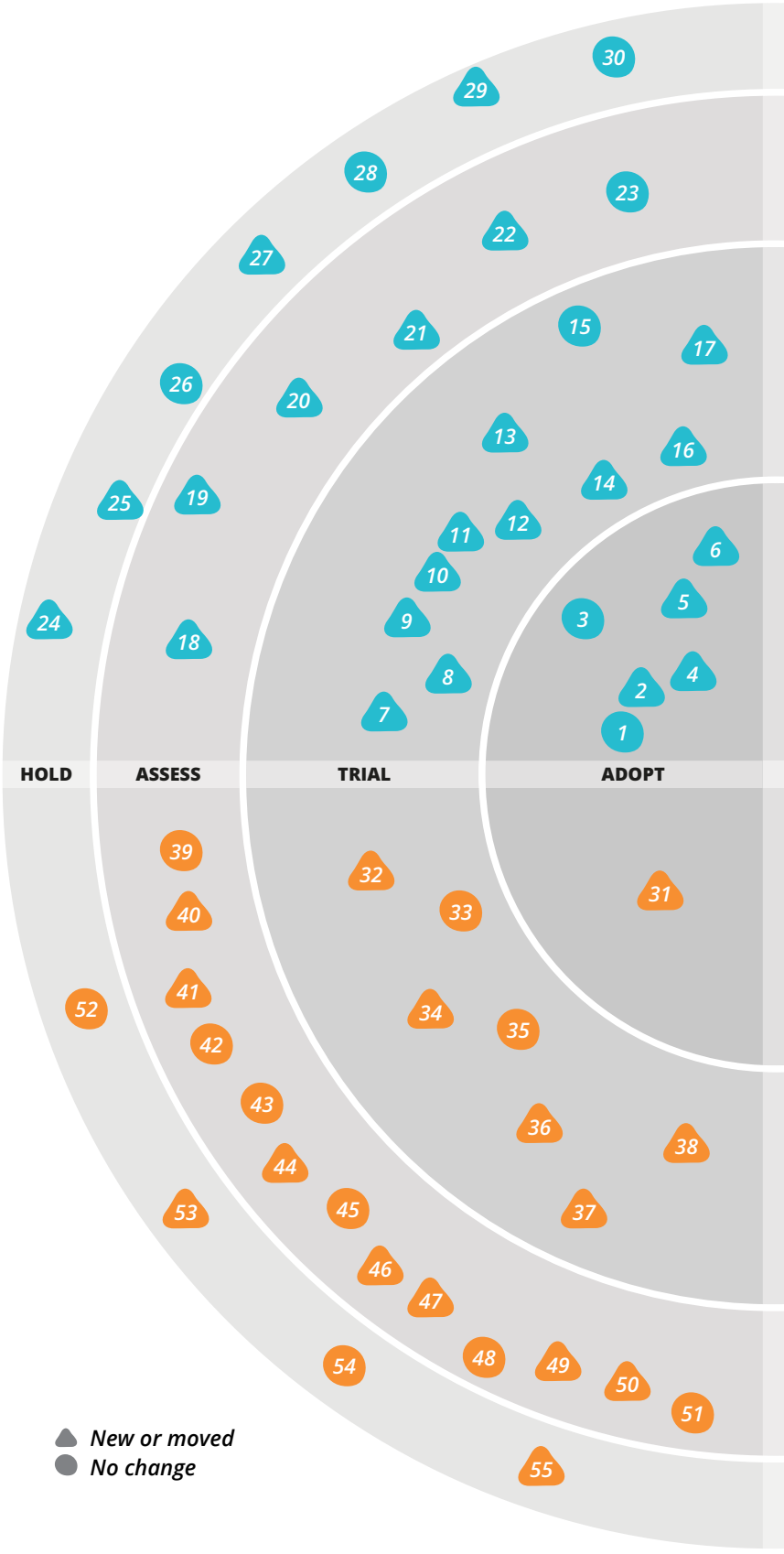
PLATFORMS

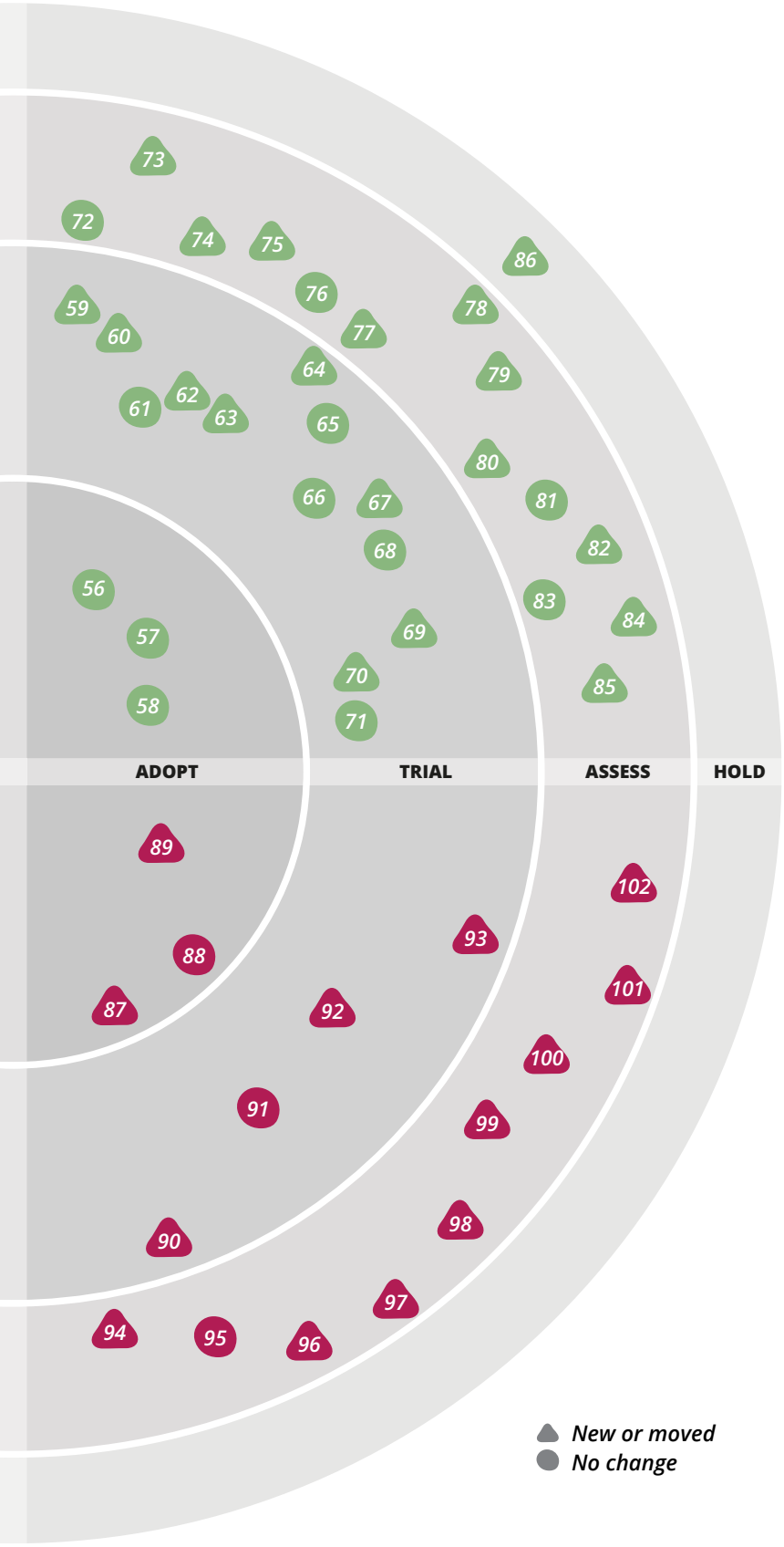
ADOPT
31. TOTP Two-Factor Authentication

TRIAL
32. Apache Mesos
33. Apache Spark
34. AWS Lambda new
35. Cloudera Impala
36. Fastly new
37. H2O
38. HSTS new

ASSESS
39. Apache Kylin
40. AWS ECS new
41. Ceph new
42. CoreCLR and CoreFX
43. Deis
44. Kubernetes new
45. Linux security modules
46. Mesosphere DCOS new
47. Microsoft Nano Server new
48. Particle Photon/Particle Electron
49. Presto new
50. Rancher new
51. Time series databases

HOLD
52. Application Servers
53. Over-ambitious API Gateways new
54. SPDY
55. Superficial private cloud new





THE RADAR

TOOLS

ADOPT

56. Composer
57. Mountebank
58. Postman

TRIAL

59. Browsersync new
60. Carthage new
61. Consul
62. Docker Toolbox new
63. Gitrob new
64. GitUp new
65. Hamms
66. IndexedDB
67. Polly
68. REST-assured
69. Sensu
70. SysDig new
71. ZAP

ASSESS

72. Apache Kafka
73. Concourse CI new
74. Espresso new
75. Gauge new
76. Gor
77. ievms new
78. Let's Encrypt new
79. Pageify new
80. Prometheus
81. Quick
82. RAML new
83. Security Monkey
84. Sleepy Puppy new
85. Visual Studio Code new

HOLD

86. Citrix for development

LANGUAGES & FRAMEWORKS

ADOPT

87. ECMAScript 6 new
88. Nancy
89. Swift

TRIAL

90. Enlive new
91. React.js
92. SignalR new
93. Spring Boot

ASSESS

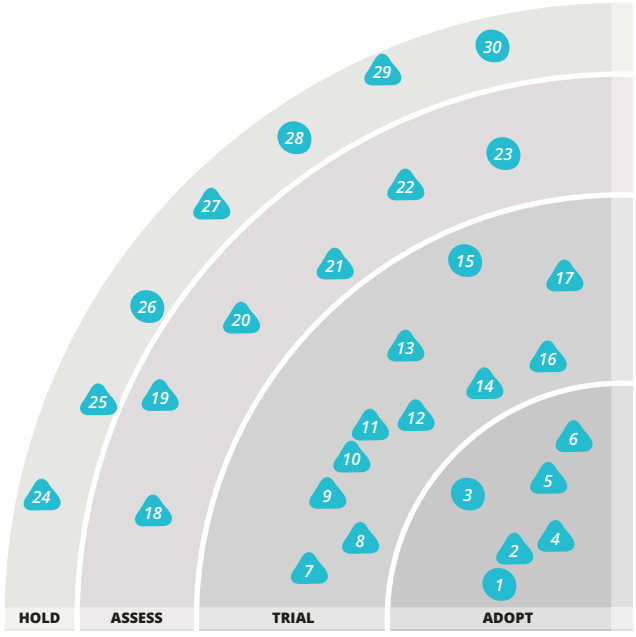
94. Axon new
95. Ember.js
96. Frege new
97. HyperResource new
98. Material UI new
99. OkHttp new
100. React Native new
101. TLA+ new
102. Traveling Ruby new

HOLD

TECHNIQUES

Implementing Continuous Delivery continues to be a challenge for many organizations, and it remains important to highlight useful techniques such as **decoupling deployment from release**. We recommend strictly using the term Deployment when referring to the act of deploying a change to application components or infrastructure. The term Release should be used when a feature change is released to end users, with a business impact. Using techniques such as feature toggles and dark launches, we can deploy changes to production systems more frequently without releasing features. More-frequent deployments reduce the risk associated with change, while business stakeholders retain control over when features are released to end users.

'Just In Time Design' is an important and useful concept for visual design that the **NoPSD** movement attempts to capture. You don't need to design the whole application or every UI element up front. Design things as you need them with as lightweight tools as you can use. We have seen a corresponding growth in simpler tools with faster learning curves, such as **Sketch**, as well as an increasing return to pen-and-paper (especially when paired with an existing robust **digital style guide**). Because of the limitations of flat mock-ups when you're designing for screens, creating prototypes of varying fidelity with tools such as **Invision**, **FramerJS** and **Origami** - or simply HTML/CSS and a bit of JavaScript - has also become increasingly common and valuable for communicating design intent.



We've long been championing the idea that thinking of software development as a project - something budgeted and delivered during a limited time slot - doesn't fit the needs of the modern business. Important software efforts need to be an ongoing product that supports and rethinks the business process it is supporting. Such efforts are not complete until the business process, and its software, cease to be useful. Our observation of this **products over projects** approach, both with our own projects and outside, makes us determine that it is the approach to use for all but exceptional cases.

With the number of high-profile security breaches in the past months, software development teams no longer need convincing that they must place an emphasis on writing secure software and dealing with their users' data in a responsible way. The teams face a steep learning curve, though, and the vast number of potential threats - ranging from organized crime and government spying to teenagers who attack systems 'for the lulz' can be overwhelming. **Threat Modelling** provides a set of techniques, mostly from a defensive perspective, that help you understand and classify potential threats. Turned into 'evil-user stories', threat models can give a team a manageable and effective approach to making their systems more secure.

ADOPT

1. Consumer-driven contract testing
2. Decoupling deployment from release
3. Generated infrastructure diagrams
4. NoPSD
5. Products over projects
6. Threat Modelling

TRIAL

7. BEM
8. BFF - Backend for frontends
9. Docker for builds
10. Event Storming
11. Flux
12. Idempotency filter
13. iFrames for sandboxing
14. NPM for all the things
15. Offline first web applications
16. Phoenix Environments
17. QA in production

ASSESS

18. Accumulate-only data
19. Bug bounties
20. Data Lake
21. Hosted IDE's
22. Monitoring of invariants
23. Reactive Architectures

HOLD

24. Gitflow
25. High performance envy/web scale envy
26. Microservice envy
27. Pace-layered Application Strategy
28. Programming in your CI/CD tool
29. SAFe™
30. Separate DevOps team

TECHNIQUES *continued*

Debugging CSS problems can be painful. How many times have you had to trawl through thousands of overridden styles to work out the source of your problem? This has led many of our teams to introduce various guidelines such as avoiding cascading and overrides, making styles opt-in and emphasizing thoughtful naming. **BEM** is a simple CSS naming convention (standing for Block, Element, Modifier) that helps give semantic clarity and structure to your CSS. By using BEM, it becomes much easier to understand which CSS rules are influencing the appearance of an element and, more importantly, the intent of those rules. This approach can be seen as moving the OO lesson of favoring composition over inheritance to the world of CSS.

Valuable services support many variations in clients, such as mobile versus web and different forms of web interface. It's tempting to design a single back-end API to support all clients with a reusable API. But client needs vary, as do constraints such as bandwidth for mobile devices versus the desire for lots of data on fast web connections. Consequently it's often best to **define different back-end services for each kind of front-end** client. These back ends should be developed by teams aligned with each front end to ensure that each back end properly meets the needs of its client.

One of the many innovative uses of Docker that we've seen on our projects is a technique to manage build-time dependencies. In the past, it was common to run build agents on an OS, augmented with dependencies needed for the target build. But with Docker it is possible to run the compilation step in an isolated environment complete with dependencies without contaminating the build agent. This technique of using **Docker for builds** has proven particularly useful for compiling Golang binaries, and the golang-builder container is available for this very purpose.

Event Storming is a useful way to do rapid "outside-in" domain modeling: starting with the events that occur in the domain rather than a static data model. Run as a facilitated workshop, it focuses on discovering key domain events, placing them along a timeline, identifying their triggers and then exploring their relationships. This approach is particularly useful for people taking a CQRS or Event Sourcing approach. Getting the right people in the room is important - a blend of business and technical people who bring both the questions and the answers. Ensuring that you have enough wall space for modeling is the second key to success. Look to discover the big picture, with the goal of collectively understanding the domain in all of its complexity, before diving into solutions.

Flux is an application architecture introduced by Facebook. Usually mentioned in conjunction with **React.js**, Flux is based on a one-way flow of data up through the rendering pipeline. Flux embraces the modern web landscape of client-side JavaScript applications in a way that avoids the venerable MV* clichés. ThoughtWorks teams are now starting to gain some experience with this architectural style and find that it meshes well with service orientation and solves some of the problems inherent in two-way data binding.

Many services, especially legacy services, are written with the assumption that any request will occur only once. Networks being what they are, this can be difficult to arrange. An **idempotency filter** is a simple component that merely checks for duplicate requests and ensures that they are sent to the supplier service only once. Such a filter should do only this one task and be used as a decorator over existing service calls.

Modern web pages tend to contain a plethora of JavaScript widgets and snippets coming from a variety of third-party sources. This can have a negative impact on both security and performance. While we are still waiting for fuller JavaScript isolation with web components, our teams have benefited from using HTML5 **iFrames for sandboxing** untrusted JavaScript.

The JavaScript world has a plethora of dependency and package-management tools, all of which rely on the Node Package Manager (NPM). Teams are starting to see these extra tools as redundant and are recommending that if you can use solely NPM for package and dependency management, you should. The simplification of using **NPM for all the things** helps reduce some of the churn in the JavaScript tools space.

TECHNIQUES *continued*

The time taken to provision and update environments continues to be a significant bottleneck on many software projects. Phoenix Environments can help with this delay by extending the idea of Phoenix Servers to cover entire environments. We feel this is such a valuable and time-saving technique that you should consider trialing this approach. Using automation, we can create whole environments - including network configuration, load balancing and firewall ports - for example by using **CloudFormation** in AWS. We can then prove that the process works by tearing the environments down and recreating them from scratch on a regular basis. **Phoenix Environments** can support provisioning new environments for testing, development, UAT and disaster recovery. As with Phoenix Servers, this pattern is not always applicable, and we need to think carefully about things like state and dependencies. Treating the whole environment as a blue/green deployment can be one approach when environment reconfiguration needs to be done.

Traditionally, QA roles have focused on assessing the quality of a software product in a pre-production environment. With the rise of Continuous Delivery, the QA role is shifting to include analyzing software product quality in production. This involves monitoring of the production systems, coming up with alert conditions to detect urgent errors, determining ongoing quality issues and figuring out what measurements you can use in the production environment to make this work. While there is a danger that some organizations will go too far and neglect pre-production QA, our experience shows that **QA in production** is a valuable tool for organizations that have already progressed to a reasonable degree of Continuous Delivery.

Immutable data structures are becoming more popular, with functional languages such as Clojure and Scala providing immutability by default. Immutability allows code to be more easily written, read and reasoned about. Using an **accumulate-only data store** can confer some of these benefits in the database layer, as well as make audit and historical querying simple. Implementation options vary, from specific accumulative data stores such as Datomic to simply using an "append-don't-update" approach with a traditional database. **Accumulate-only** is a design strategy whereby data is removed via retraction rather than update; **append-only** is an implementation technique.

More and more organizations are starting to use **bug bounties** to encourage reporting of what are often security-related bugs, and in general help improve the quality of their software. To support these programs, companies like HackerOne and BugCrowd can help organizations manage this process more easily. We have limited experience with these offerings ourselves, but we like the idea of encouraging people to help come forward and highlight what can often be damaging vulnerabilities in an open and transparent way. It's worth noting that there might be some legal issues with encouraging users to find vulnerabilities in your software, so please do check that out first.

A **Data Lake** is an immutable data store of largely unprocessed 'raw' data, acting as a source for data analytics. Whereas the more familiar Data Warehouse filters and processes the data before storing it, the lake just captures the raw data, leaving it to the users of that data to carry out the particular analysis that they need. Examples include HDFS or HBase within a Hadoop, Spark or Storm processing framework. Usually only a small group of data scientists work on the raw data, developing streams of processed data into lakeshore data marts for most users to query. A Data Lake should only be used for analytics and reporting. For collaboration between operational systems we prefer using services designed for that purpose.

Many organizations want to leverage distributed or offshore development but have security concerns with their code and other intellectual property sitting outside their control. The result is often to use high-latency remote-desktop solutions for development, adhering to an organization's security controls but crippling developer productivity. An alternative is to use a **Hosted IDE** delivered to a browser via VPN. The IDE, code and build environment are hosted within the organization's private cloud, easing security concerns, and the developer experience is significantly improved. Tools in this space include Orion and Che from the Eclipse Foundation, Cloud9 and Code Envy.

In monitoring, the common approach is to conceive of erroneous conditions and set alerts when these appear. But it's often difficult to enumerate the myriad failure modes in a software system. **Monitoring of invariants** is a complementary approach to setting expected normal ranges, often by examining historical behavior, and alerting whenever a system goes outside those bounds.

TECHNIQUES *continued*

We firmly believe that long-lived version-control branches harm valuable engineering practices such as continuous integration, and this belief underlies our dislike for **Gitflow**. We love the flexibility of Git underneath but abhor tools that encourage bad engineering practices. Very short-lived branches hurt less, but most teams we see using Gitflow feel empowered to abuse its branch-heavy workflow, which encourages late integration (therefore discouraging true continuous integration).

We see many teams run into trouble because they have chosen complex tools, frameworks or architectures because they 'might need to scale'. Companies such as Twitter and Netflix need to be able to support extreme loads and so need these architectures, but they also have extremely skilled development teams able to handle the complexity. Most situations do not require these kinds of engineering feats; teams should keep their **web scale envy** in check in favor of simpler solutions that still get the job done.

Gartner's **Pace-layered Application Strategy** approach appears to be creating an unhelpful focus on the idea of layers within an architecture. We find thinking about the pace of change within different **business capabilities** (which can be made up of several architectural layers) to be a more useful concept. The danger in focusing on layers is that many types of change cut across multiple layers. For example, being able to add new class of stock to a website is not just about having an easy-to-change CMS; you also need to update the database, integration points, warehouse systems, etc. The recognition that some parts of an architecture need to be more maneuverable than others is useful. However, a focus on layers is proving unhelpful.

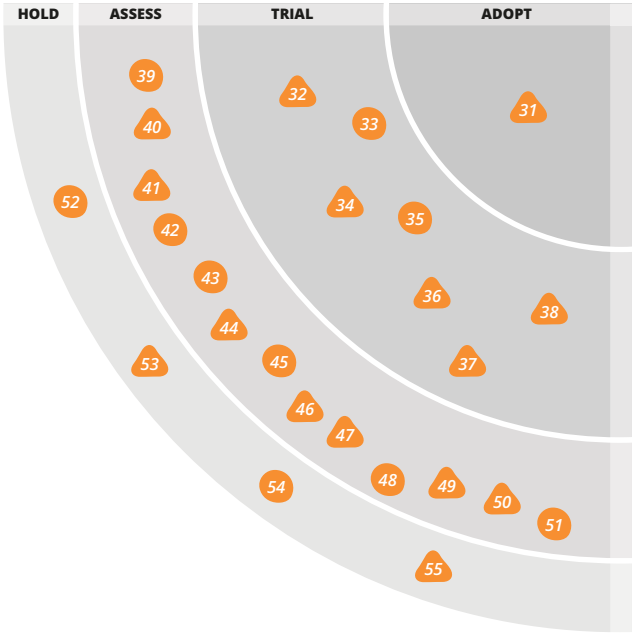
The Scaled Agile Framework® (aka **SAFe™**) continues to gain mindshare in many organizations at scale. In addition, tools and certification are becoming a significant aspect of the adoption of SAFe™. We continue to be concerned that actual adoptions are prone to over-standardization and are tending towards large release practices, resulting in practices that hinder agile adoption. In its place, we continue to recommend lean approaches that include experimentation and incorporate continuous improvement practices like the Improvement Katas offer organizations a better model for scaling agile.

Scaled Agile Framework® and SAFe™ are trademarks of Scaled Agile, Inc.

PLATFORMS

Password security is still a hotly debated topic with the UK government advocating technical controls that let users remember simpler passwords and Edward Snowden's password advice being described as only 'borderline secure'. Passwords are generally one of the weakest links in the security chain, so we recommend employing **two-factor authentication**, which can significantly improve security. Time-based One-Time Password (**TOTP**) is the standard algorithm in this space, with straightforward server-side implementations and free smartphone authenticator apps from Google and Microsoft.

Mesos is a platform that abstracts out underlying computing resources to make it easier to build massively scalable distributed systems. It can be used to provide a scheduling layer for Docker, or to act as an abstraction layer to things like AWS. Twitter has used it to great effect to help it scale its infrastructure. Tools built on top of Mesos are starting to appear, such as Chronos, which is a distributed, fault-tolerant cron replacement. Prominent success stories are appearing, such as Apple's Siri rearchitecting to use Mesos.



AWS releases a huge number of new features on what seems like a monthly basis, so it can sometimes be hard for any new service offering to rise above the noise, but **Lambda** certainly manages to attract notice. Initially just supporting JavaScript, but now adding support for JVM-based applications (with more no doubt to follow), Lambda allows you to fire up very short-lived processes either in reaction to an event, or via a call from the related API Gateway. For stateless services, this means you don't need to worry about running any long-lived machines, potentially reducing costs and improving security. Despite other forays into the PaaS space by AWS, Lambda looks the closest to getting this right.

Fastly, one of a number of CDNs on the market, has a large and growing following on ThoughtWorks projects and is used by many web-scale household names, such as GitHub and Twitter. Its feature set, speed and price point combine to make it a very attractive option when you're looking for an edge caching solution. We have also seen significant cost savings on projects that move to this platform from another CDN. If you are in the market for a CDN, you could do worse than investigate this one.

Predictive analytics are used in more and more products, often directly in end user-facing functionality. **H2O** is an interesting open source package (with a startup behind it) that makes predictive analytics accessible to development teams, offering straightforward use of a wide variety of analytics, great performance and easy integration on JVM-based platforms. At the same time it integrates with the data scientists' favorite tools, R and Python, as well as Hadoop and Spark.

ADOPT
31. TOTP Two-Factor Authentication

TRIAL
32. Apache Mesos
33. Apache Spark
34. AWS Lambda
35. Cloudera Impala
36. Fastly
37. H2O
38. HSTS

ASSESS
39. Apache Kylin
40. AWS ECS
41. Ceph
42. CoreCLR and CoreFX
43. Deis
44. Kubernetes
45. Linux security modules
46. Mesosphere DCOS
47. Microsoft Nano Server
48. Particle Photon/Particle Electron
49. Presto
50. Rancher
51. Time series databases

HOLD
52. Application Servers
53. Over-ambitious API Gateways
54. SPDY
55. Superficial private cloud

PLATFORMS *continued*

HTTP Strict Transport Security (**HSTS**) is a now widely supported policy that allows websites to protect themselves from downgrade attacks. A downgrade attack in the context of HTTPS is one that can cause users of your site to fall back to HTTP rather than HTTPS, allowing for further attacks such as man-in-the-middle attacks. By using the server header, you inform browsers that they should only use HTTPS to access your website, and should ignore downgrade attempts to contact the site via HTTP. Browser support is now widespread enough that this easy-to-implement feature should be considered for any site using HTTPS.

The **Elastic Container Service (ECS)** is AWS' entry into the multihost Docker space. Although there is a lot of competition in this area, there aren't many off-premises managed solutions out there yet. Although ECS seems like a good first step, we are worried that it is overly complicated at the moment and lacks a good abstraction layer. If you want to run Docker on AWS, though, this tool should certainly be high on your list. Just don't expect it to be easy to get started with.

Ceph is a storage platform that can be used as object storage, as block storage, and as a file system, typically running on a cluster of commodity servers. With its first major release having been in July 2012, Ceph is certainly not a new product. We do want to highlight it on this Technology Radar as an important building block for private clouds. It is particularly attractive because its RADOS Gateway component can expose the object store through a RESTful interface that is compatible with Amazon S3 and the OpenStack Swift APIs.

Kubernetes is Google's answer to the problem of deploying containers into a cluster of machines, which is becoming an increasingly common scenario. It is not the solution used by Google internally, but an open-source project that originated at Google and has seen a fair share of external contributions. Docker and Rocket are supported as container formats and services offered include health management, replication, and discovery. A similar solution in this space is **Rancher**, an open-source solution that also allows deployment of containers into a cluster of machines. It provides services such lifecycle management, monitoring, health checks, and discovery. Also included is a completely containerized operating system based on Docker. The broad focus on containerization and very small footprint are key advantages for Rancher.

Mesosphere DCOS is a platform built on top of Mesos. It provides an abstraction over underling machines, giving you a pool of storage and compute that allows services built for DCOS to operate at massive scale (Support is already there for Hadoop, Spark and Cassandra, among others). This is probably overkill for more modest workloads at the moment (where plain old Mesos could still be a good fit), but it will be interesting to see if Mesosphere starts trying to position DCOS as a general-purpose system.

In contrast to modern cloud and container solutions based on Linux, even Windows Server Core is large and unwieldy. Microsoft is reacting and has provided the first previews of **Nano Server**, a further-stripped-down version of Windows Server that drops the GUI stack, 32-bit Win32 support, local logins and remote desktop support, resulting in an on-disk size of about 400MB. The early previews are difficult to work with, and the final solution will be restricted to using the CoreCLR, but for companies that are interested in running .NET-based solutions, Nano Server is definitely worth a look at this stage.

Presto is an open source distributed SQL query engine designed and optimized for running interactive analytics workloads. Presto's massively parallel processing architecture - combined with advanced code-generation techniques and in-memory processing pipelines - makes it highly scalable. It supports a large subset of ANSI SQL including complex queries, joins, aggregations and window functions. Presto comes with support for a wide range of data sources including **Hive**, **Cassandra**, **MySQL** and **PostgreSQL**, thereby unifying the interactive analytics interface across data stores of an organization. Applications can connect to Presto using its JDBC interface.

One of our common complaints is the pushing of business smarts into middleware, resulting in application servers and enterprise service buses with ambitions to run critical application logic. These require complex programming in environments not well suited to the purpose. We're seeing a worrying re-emergence of this disease with **overambitious API Gateway** products. API Gateways can provide utility in dealing with some generic concerns - for example, authentication and rate-limiting - but any domain smarts such as data transformation or rule processing should live in applications or services where they can be controlled by product teams working closely with the domains they support.

PLATFORMS *continued*

We've seen the indisputable productivity gains that come from deployment of applications and services into mature cloud providers. Much of that gain comes from the ability of teams to deploy and operate their own services with a high degree of autonomy and responsibility. We are now regularly coming across **Superficial Private Cloud** offerings within organizations, where basic virtualization platforms are being given the "cloud" label. Often teams can self-provision a restricted set of fixed service types with limited access and little ability to customize the centrally governed "enterprise blueprints," leading to kludge solutions. Deployment pace regularly remains constrained by manually provisioned infrastructure such as network, firewall and storage. We encourage organizations to more fully consider the costs of mandating the use of an inadequate private cloud offering.

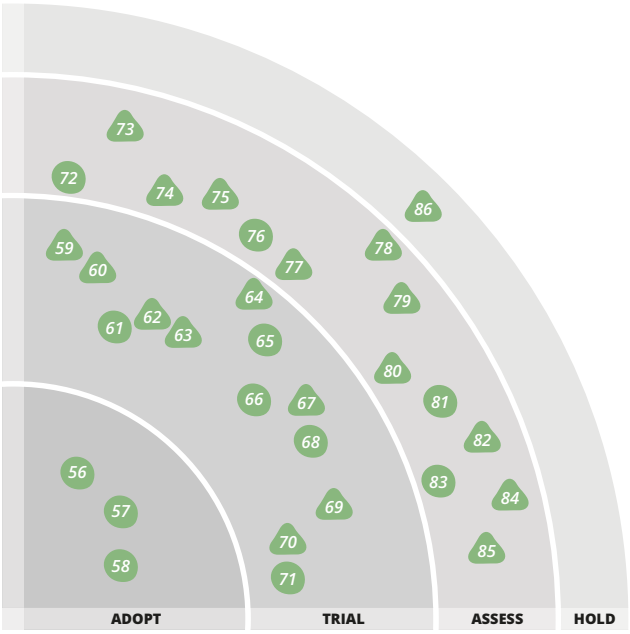
TOOLS

We've had rave reviews from a number of ThoughtWorks teams using **Browsersync**. As the number of devices we deliver web applications to grows, so does the amount of effort that must be devoted to testing across these different devices. Browsersync is a free, open source tool that can dramatically reduce this effort by synchronizing manual browser testing across multiple mobile or desktop browsers. Providing both a CLI and a UI option, the tool is build-pipeline friendly and automates repetitive tasks such as form filling.

Dependency management in iOS and OS X projects used to be either completely manual or completely automatic as part of using CocoaPods. With **Carthage**, a new middle ground has become available. Carthage manages dependencies - it downloads, builds and updates frameworks - but it leaves the integration of the frameworks into the build of the project to the project. This is in contrast to CocoaPods, which basically takes over the project structure and build setup. It should be noted that Carthage can only deal with dynamic frameworks, which are not available on iOS 7 and below.

Previously, we recommended boot2docker as a way of easily running Docker on your local Windows or OS X machine. **Docker Toolbox** now replaces boot2docker, adding some tooling as well. Now included is Kitematic for managing your containers, as well as Docker Compose for managing multi-Docker setup (Mac only). It can be used safely as a drop-in replacement for boot2docker, and it will even handle the upgrade for you.

Safely storing secrets such as passwords and access tokens in code repositories is now supported by a growing number of tools - for example, git-crypt and Blackbox, which we mentioned in the previous



Technology Radar. Despite the availability of these tools, it is still, unfortunately, all too common that secrets are stored unprotected. In fact, it is so common that automated exploit software is used to find AWS credentials and spin up EC2 instances to mine Bitcoins, leaving the attacker with the Bitcoins and the account owner with the bill. **Gitrob** takes a similar approach and scans an organization's GitHub repositories, flagging all files that might contain sensitive information that shouldn't have been pushed to the repository. This is obviously a reactive approach. Gitrob can only alert teams when it is (almost) too late. For this reason, Gitrob can only ever be a complementary tool, to minimize damage.

ADOPT
56. Composer
57. Mountebank
58. Postman

TRIAL
59. Browsersync
60. Carthage
61. Consul
62. Docker Toolbox
63. Gitrob
64. GitUp
65. Hamms
66. IndexedDB
67. Polly
68. REST-assured
69. Sensu
70. SysDig
71. ZAP

ASSESS
72. Apache Kafka
73. Concourse CI
74. Espresso
75. Gauge
76. Gor
77. ievms
78. Let's Encrypt
79. Pageify
80. Prometheus
81. Quick
82. RAML
83. Security Monkey
84. Sleepy Puppy
85. Visual Studio Code

HOLD
86. Citrix for development

TOOLS *continued*

Git can be confusing. Really confusing. And even when it's used in a simple trunk-based development process, there are still enough nuances to how it works that people can tie themselves in knots from time to time. When this happens, having an understanding of how Git works under the hood is very useful, and **GitUp** is a Mac-based tool that gives you exactly that. GitUp provides a graphical representation of what is happening as you type normal Git commands into the terminal. You can learn the various Git commands while also understanding what each one does as you use it. GitUp is a useful tool for both people new to Git and those with more Git experience.

Several of our teams working on .NET projects have recommended **Polly** as being useful in building microservice-based systems. It encourages the fluent expression of transient exception-handling policies and the Circuit Breaker pattern, including policies such as Retry, Retry Forever and Wait and Retry. Similar libraries already exist in other languages (Hystrix for Java for example), and Polly is a welcome addition from the .NET community. Integrating well with Polly is **Brighter**. Brighter is another small open source .Net library that provides scaffolding to implement command invocation. Combining the two libraries provides useful circuit-breaking functionality especially in the context of the Ports and Adapters pattern and CQRS. Although they can be used separately, in the wild our teams find they work well together.

Many monitoring tools are built around the concept of the machine or instance. The increasing use of patterns like Phoenix Server and tools like Docker mean this is an increasingly unhelpful way to model infrastructure: Instances are becoming transient while services are the things that persist. **Sensu** allows an instance to register itself as playing a particular role, and Sensu then monitors it on that basis. Over time, different instances playing that role may come and go. Given these factors and the increasing maturity of the tool, we felt it was time to bring Sensu back on to the radar.

Although **SysDig** isn't the newest tool on the Technology Radar, we're still surprised by how many people haven't heard of it. A pluggable open source CLI for Linux system troubleshooting, SysDig has some pretty powerful features. One of the key things we like is the ability to generate a system trace on a machine that is experiencing difficulties, which you can then interrogate afterward to find out what was happening. SysDig also contains support for working with containers, something that makes a previously useful tool even more powerful.

Many development teams are making the move from simple continuous integration servers to Continuous Delivery pipelines, often spanning multiple environments, reaching into production. To implement such a pipeline successfully and operate it in a sustainable way requires a CI/CD tool that treats build pipelines and artifacts as first-class citizens; and unfortunately there aren't many. **Concourse CI** is a promising new entrant in this field, and our teams that have tried it are excited about its setup, which enables builds that run in containers, has a clean, usable UI and discourages snowflake build servers.

Espresso is an Android functional-testing tool. Its small-core API hides the messy implementation details and helps in writing concise tests, with faster and reliable test execution.

Gauge is a lightweight cross-platform test-automation tool. Specifications are written in free-form Markdown, so test cases can be written in the business language and can be incorporated into any existing documentation format. Supported languages are implemented as plugins to a single core implementation, which ensures consistency across language implementations. This tool, open sourced by ThoughtWorks, also supports parallel execution out of the box for all supported platforms.

Despite the shrinking usage of Internet Explorer, for many products the IE user base is not an insignificant share of the market, and browser compatibility needs to be tested. This is particularly troublesome if you prefer the joys of a UNIX-based system for development. To aid in this dilemma, **ievms** provides a utility script that brings together Windows-distributed VM images and VirtualBox to automate the setup and testability of various IE versions, from 6 up to Edge.

TOOLS *continued*

Although more sites every day are implementing HTTPS to help protect their own users and improve the integrity of the web as a whole, there are many more sites to go. In addition, we see more and more people using HTTPS within their enterprises, to provide additional security guarantees. One of the main blockers to wider adoption has been the process of getting a certificate in the first place. Aside from the cost, the process itself is far from slick. **Let's Encrypt**, a new Certificate Authority, aims to solve all this. First, it provides certificates for free. Second, and arguably more important, it also provides an extremely easy-to-use command-line API, making it easy to fully automate the process of issuing, upgrading and installing certificates. We think that Let's Encrypt, in beta at the moment, has the chance to be revolutionary in terms of helping more of the web get on to HTTPS, and at the same time showing what good, automatable tools for the security-conscious should look like.

Pageify is a Ruby library for building page objects for UI automation tests, focusing on faster test execution and code readability. It offers simple APIs to dynamically define, operate and assert on the page objects, allowing readable code even when handling elements with complex hierarchies in the DOM. It bundles integration for **WebDriver** and **Capybara**.

SoundCloud has recently open sourced its monitoring and alerting toolkit, **Prometheus**. Developed in reaction to difficulties with Graphite in its production systems, Prometheus primarily supports a pull-based HTTP model (although a more Graphite-like push model is also supported). It also goes further by supporting alerts, making it an active part of your operational toolset. As of this writing, Prometheus is still only in release 0.15.1 but is evolving rapidly. We're glad to see the recent product focus on core time-series DB and multidimensional indexing capabilities while allowing for export to a wider variety of front-end graphing tools.

With a growing landscape of services providing RESTful APIs, it is becoming increasingly important to document them. We have previously mentioned Swagger, and in this Technology Radar we'd like to highlight the RESTful API modeling language (**RAML**). Our teams feel that in comparison to Swagger it is more lightweight and moves the focus from adding documentation to existing APIs to designing APIs.

Sleepy Puppy is a delayed cross-site scripting (XSS) payload-management framework recently open sourced by Netflix. It enables you to test vulnerabilities for XSS past the target application when the perpetrator intends to attack a secondary underlying system. With XSS being one of the OWASP Top10, we see this framework assisting with automated security checks for several applications. It simplifies the capturing, managing and tracking of XSS propagation over long periods of time, with customizable payloads. Sleepy puppy also exposes an API that can be integrated with vulnerability tools like ZAP, for automated security checks.

Visual Studio Code is Microsoft's free IDE editor, available across platforms. We find the version-control integration with Git very beneficial to promoting continuous integration practices. Visual Studio Code also provides a means of integrating with external tools via tasks, with autodetection of grunt/gulp tasks eliminating the need for running grunt/gulp tasks via terminals and simply using the editor. With the growth of the Docker ecosystem, this IDE offers support for the dockerfile with snippets and definitions of valid commands.

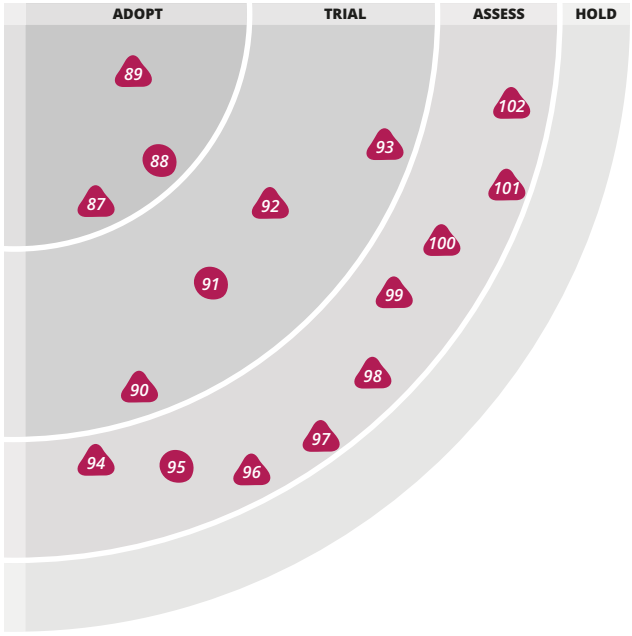
Many organizations are still forcing distributed or offshore development teams to use **Citrix remote desktop for development**. Although this provides a simple security model – assets supposedly never leave the organization's servers - using remote desktops for development absolutely cripples developer productivity. There's not much point paying a cheaper hourly rate for developers if you're going to impose both the distribution and remote-desktop burdens on them, and we wish more offshore vendors would admit these drawbacks to their clients. It's much better to use either a 'clean room' secured offshore environment where local development can be done, or a Hosted IDE (e.g. ievms).

LANGUAGES & FRAMEWORKS

Over many years, JavaScript has grown to become probably the most widely used programming language in the world. Nevertheless, the language itself has a few problems that many have attempted to address by using libraries or even by implementing their own languages that run on top of JavaScript (of which we've mentioned both CoffeeScript and ClojureScript before). **ECMAScript 6**, the new version of JavaScript, addresses many of the concerns of the older versions currently in use. Although browser support is scarce, support from mature transpilers like Babel allows you to write ECMAScript 6 and have it supported in older browsers. For new projects, we strongly suggest starting with ECMAScript 6 from day one.

A year after its public debut, **Swift** is now our default choice for development in the Apple ecosystem. With the recent release of Swift 2, the language approaches a level of maturity that provides the stability and performance required for most projects. Swift still has issues, especially around tool support, refactoring and testing. However, we feel that these are not substantial enough to warrant avoiding Swift. At the same time, porting large, existing Objective-C codebases is unlikely to pay off. The announcement that Swift will become open source software is a further positive sign. We are hopeful that this will not just be another dumping of internally developed code into a public repository, because Apple has clearly stated that community contributions are encouraged and will be accepted.

Most templating frameworks like Mustache or FreeMarker mix code with markup in a single file to implement complex, dynamic content. **Enlive** is a Clojure-based templating framework that completely separates programming language from HTML markup and employs CSS selectors to find and replace parts of the document. Enlive demonstrates the power of functional programming to implement complex behavior through a series of simple, composable functions acting on a common abstraction. Our teams working in Clojure have found it to be a very useful and straightforward tool.



We have a number of reservations about the use of HTML5 WebSockets. By allowing the server to initiate actions on the browser, WebSockets departs from the connectionless, request/response model that underpins the World Wide Web today. Security is another big risk with WebSockets. For example, the standard does not impose any cross-origin request policy. However, we do recognize that in certain monitoring or alerting applications, WebSockets can be very useful. If you need to build a .NET WebSockets server, **SignalR** conveniently implements much of the additional code you need for a robust production application. This includes some recommended security practices such as validating connection tokens and activating SSL when encryption is needed. Although ThoughtWorks teams have been very happy with SignalR, there are still fundamental issues with WebSockets that you should consider before diving in.

ADOPT
87. ECMAScript 6
88. Nancy
89. Swift

TRIAL
90. Enlive
91. React.js
92. SignalR
93. Spring Boot

ASSESS
94. Axon
95. Ember.js
96. Frege
97. HyperResource
98. Material UI
99. OkHttp
100. React Native
101. TLA+
102. Traveling Ruby

HOLD

LANGUAGES & FRAMEWORKS *continued*

Spring Boot allows easy setup of standalone Spring-based applications. It's ideal for pulling up new microservices and easy to deploy. It also makes data access less of a pain, thanks to the Hibernate mappings with much less boilerplate code. We like that Spring Boot simplifies Java services built with Spring but have learned to be cautious of the many dependencies. Spring still lurks just beneath the surface. If you're writing microservices with Java, you might also consider using **DropWizard** or a microframework like Spark to get the benefits of Spring Boot without the enormous weight of Spring.

While we still have some reservations about CQRS as a general pattern, the approach can work very well in specific places. In those specific situations, however, a lot of work is left to the developer to properly execute CQRS. **Axon** is a framework that can help with this on the JVM, and we've used it with some success. Although it certainly can't be considered a perfect solution right now, it continues to evolve and may make much more sense than trying to write everything from scratch.

Following many other programming languages, one of the language geeks' absolute favourites, Haskell, is now also available on the JVM in the form of **Frege**. This brings a purely functional programming language onto the platform, allowing for easy interoperability with other JVM languages and libraries.

HyperResource is a Ruby framework for building a RESTful API client. The framework accepts JSON in **HAL** format and dynamically generates a model object complete with hypermedia links. Although the framework is still in its infancy, we like that it embraces Richardson level 3 REST for better service discoverability and self-documenting protocols.

Material UI provides reusable components for use in React applications that implement Google's Material Design language. Filling a similar space to Twitter Bootstrap, it gets you up and running quickly but doesn't have the same drawbacks as your application grows. Elemental UI is worth investigating as an alternative.

OkHttp is a Java HTTP connection library from Square that provides a fluent interface for creating connections, as well as support for the faster HTTP/2 protocol.

Even when using HTTP/1.1, OkHttp can provide performance improvements via connection pooling and transparent gzip compression. Supporting both blocking synchronous and nonblocking asynchronous calls, it can also be used as a drop-in replacement for the widely used Apache HttpClient.

Yet another entrant into the cross-platform mobile development world, Facebook's **React Native** brings the React.js programming model to iOS and Android developers. React Native programs are written in JavaScript, but unlike a hybrid framework such as Ionic, React Native gives developers access to native UI components on the target platform. This is an approach we've seen before (e.g., Calatrava), but React Native has already inspired a substantial developer community and builds on the momentum generated by React.js. This framework could play a significant role in the future of mobile app development.

Building systems using microservices requires us to think more deeply about failure isolation and testing. **TLA+** is a formal specification language that can be useful in both these scenarios. For failure isolation, TLA+ can be used to identify invariants in your system that can be monitored directly. An invariant can be the ratio of number of requests to one service to the number of requests to a second service, for example. Any change in this ratio would lead to an alert. TLA+ is also being used to identify subtle design flaws in distributed systems. Amazon, for example, used model-checking based on a formal specification written in TLA+ to identify subtle bugs in Dynamo DB before it was released to the public. For most systems, the investment required to create the formal specification and then perform model checking is probably too great; however, for critical systems - complex ones, or those with many users - we think it's very valuable to have another tool in our toolbox.

Traveling Ruby makes it possible to distribute portable, ready-to-run, platform-agnostic Ruby binaries without the need to install an interpreter, packages or additional gems. It decouples running Ruby applications from the development environment they run in.

ThoughtWorks is a software company and community of passionate, purpose-led individuals that specialize in software consulting, delivery and products. We think disruptively to deliver technology to address our clients' toughest challenges, all while seeking to revolutionize the IT industry and create positive social change. We make pioneering tools for software teams who aspire to be great. Our products help organizations continuously improve and deliver quality software for their most critical needs. Founded over 20 years ago, ThoughtWorks has grown from a small group in Chicago to a company of over 3500 people spread across 35 offices in 12 countries: Australia, Brazil, Canada, China, Ecuador, Germany, India, Singapore, South Africa,Turkey, the United Kingdom, and the United States.

ThoughtWorks®